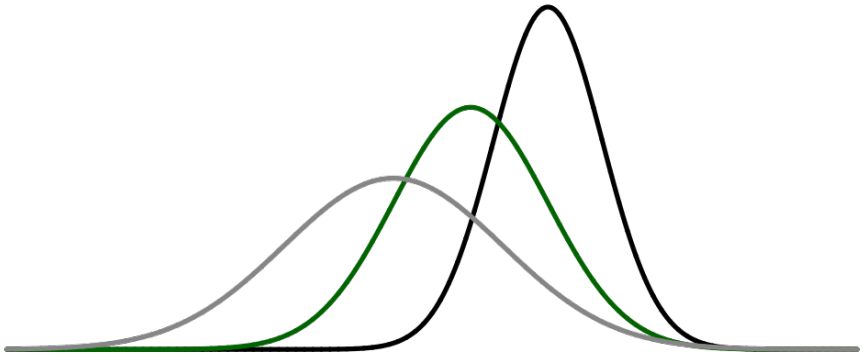
Basics of Statistical Learning

David Dalpiaz

2019-10-09

Contents

Preface



Welcome to Basics of Statistical Learning!

- TODO: Warning about development.
- TODO: Warning about PDF version.
- TODO: discuss https://daviddalpiaz.github.io/r4sl/
- TODO: course vs book
- TODO: stat432.org
- TODO: https://yihui.name/en/2013/06/fix-typo-in-documentation/
- TODO: http://varianceexplained.org/r/ds-ml-ai/

Acknowledgements

The following is a (likely incomplete) list of helpful contributors.

- Jae-Ho Lee - STAT 432, Fall 2019

Figure 1: CC NC SA

- TODO: Transfer acknowledgements from R4SL

License

Chapter 1

Introduction

```
library(readr)
library(tibble)
library(dplyr)
library(purrr)
library(ggplot2)
library(ggridges)
library(lubridate)
library(randomForest)
library(rpart)
library(rpart.plot)
library(cluster)
library(caret)
library(factoextra)
library(rsample)
library(janitor)
library(rvest)
library(dendextend)
library(knitr)
library(kableExtra)
library(ggthemes)
```

- TODO: Show package messaging? check conflicts!
- TODO: Should this be split into three analyses with different packages?

1.1 Regression: Powerlifting

1.1.1 Background

- TODO: https://www.openpowerlifting.org/
- TODO: https://en.wikipedia.org/wiki/Powerlifting

1.1.2 Data

- TODO: Why readr::col_factor() and not just col_factor()?
- TODO: Characters should be character and "categories" should be factors.
- TODO: Is `na.omit()` actually a good idea?

```
pl = read_csv("data/pl.csv", col_types = cols(Sex = readr::col_factor()))
```

```
pl
```

```
## # A tibble: 3,604 x 8
##    Name             Sex   Bodyweight   Age Squat Bench Deadlift Total
##    <chr>            <fct>      <dbl> <dbl> <dbl> <dbl>    <dbl> <dbl>
##  1 Ariel Stier      F           60      32 128.   72.5     150   350
##  2 Nicole Bueno     F           60      26 110    60       135   305
##  3 Lisa Peterson    F           67.5    28 118.   67.5     138.  322.
##  4 Shelby Bandula   F           67.5    26  92.5  67.5     140   300
##  5 Lisa Lindhorst   F           67.5    28  92.5  62.5     132.  288.
##  6 Laura Burnett    F           67.5    30  90    45       108.  242.
##  7 Suzette Bradley  F           75      38 125    75       158.  358.
##  8 Norma Romero     F           75      20  92.5  42.5     125   260
##  9 Georgia Andrews  F           82.5    29 108.   52.5     120   280
## 10 Christal Bundang F           90      30 100    55       125   280
## # ... with 3,594 more rows
```

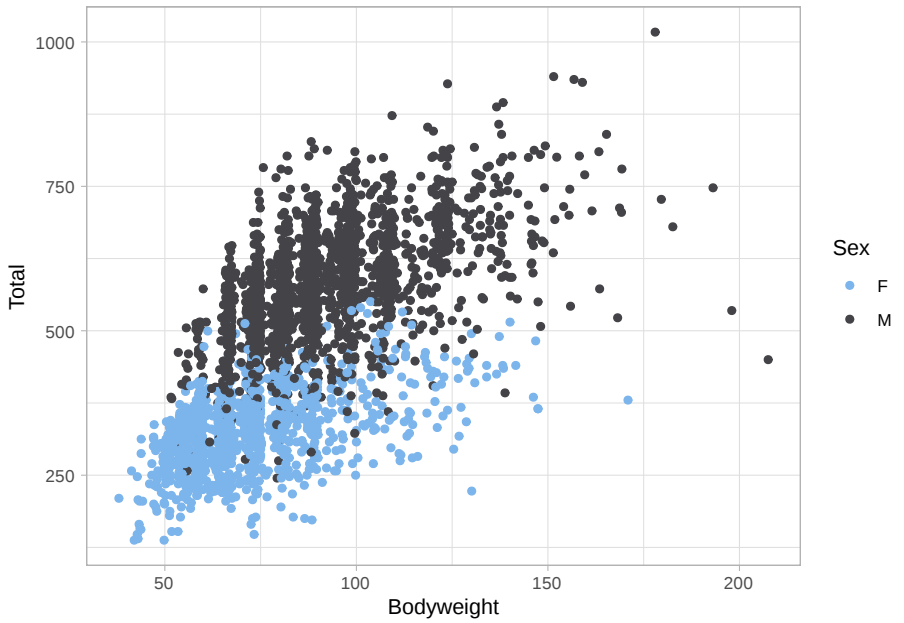
1.1.3 EDA

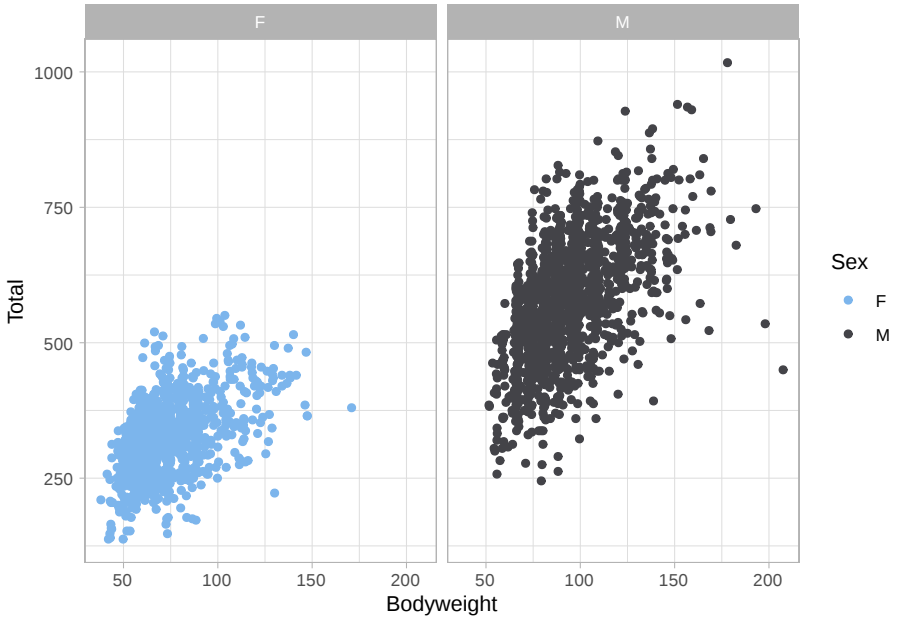
```
set.seed(1)

# test-train split
pl_tst_trn_split = initial_split(pl, prop = 0.80)
pl_trn = training(pl_tst_trn_split)
pl_tst = testing(pl_tst_trn_split)

# estimation-validation split
pl_est_val_split = initial_split(pl_trn, prop = 0.80)
```

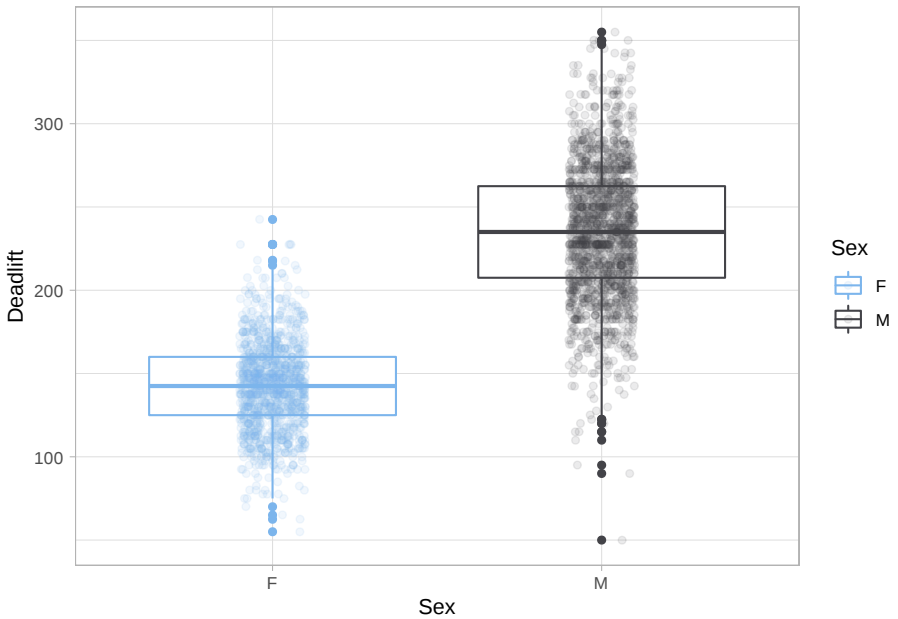
```
pl_est = training(pl_est_val_split)
pl_val = testing(pl_est_val_split)
```

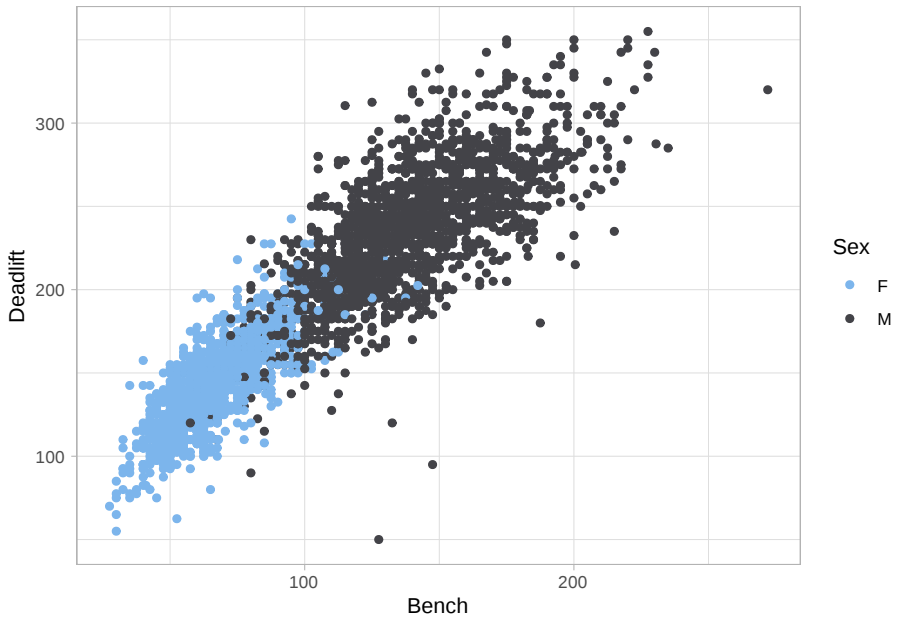
```
rm(pl)
```

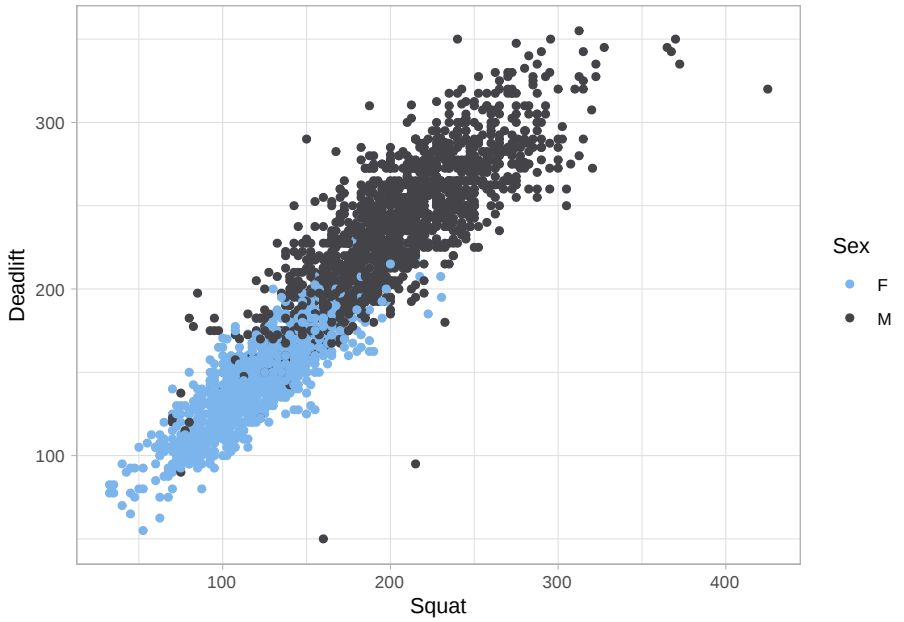
- TODO: Train can be used however you want. (Including EDA.)
- TODO: Test can only be used after all model decisions have been made!




F
M
1000
750
500
250
Total
50
100
150
200
Bodyweight
Sex
F
M


300
200
100
Deadlift
F
M
Sex
Sex
F
M

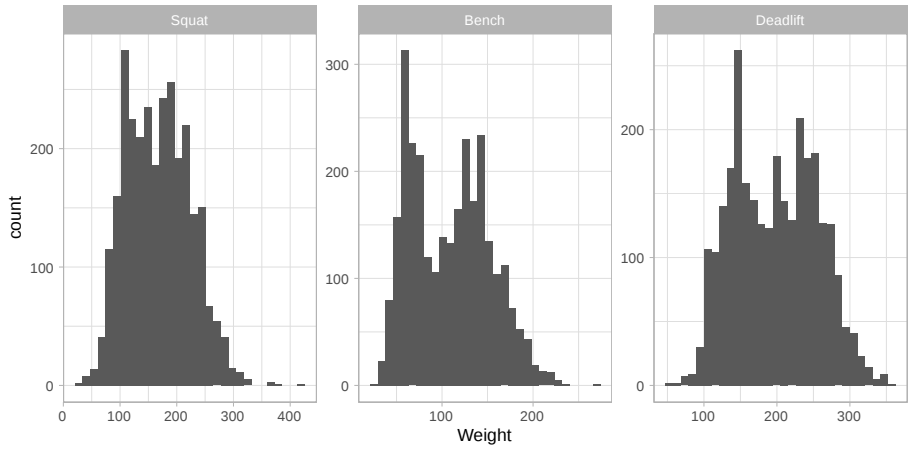


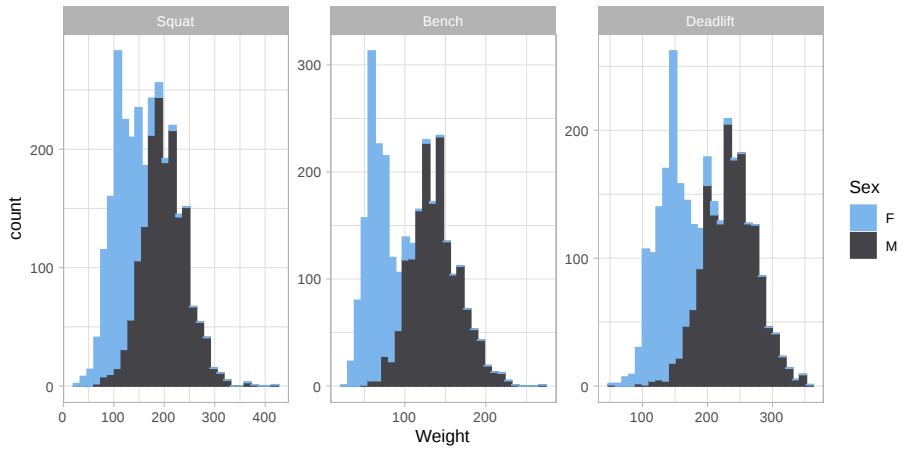


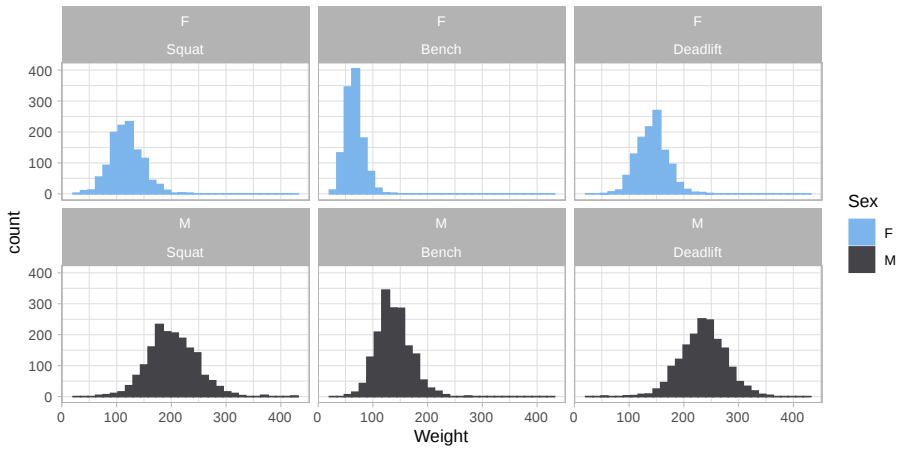

```
pl_trn_tidy = gather(pl_trn, key = "Lift", value = "Weight",
                     Squat, Bench, Deadlift)
```

```
pl_trn_tidy$Lift = factor(pl_trn_tidy$Lift, levels = c("Squat", "Bench", "Deadlift"))
```

- TODO: https://www.tidyverse.org/
- TODO: https://en.wikipedia.org/wiki/Tidy_data
- TODO: http://vita.had.co.nz/papers/tidy-data.pdf





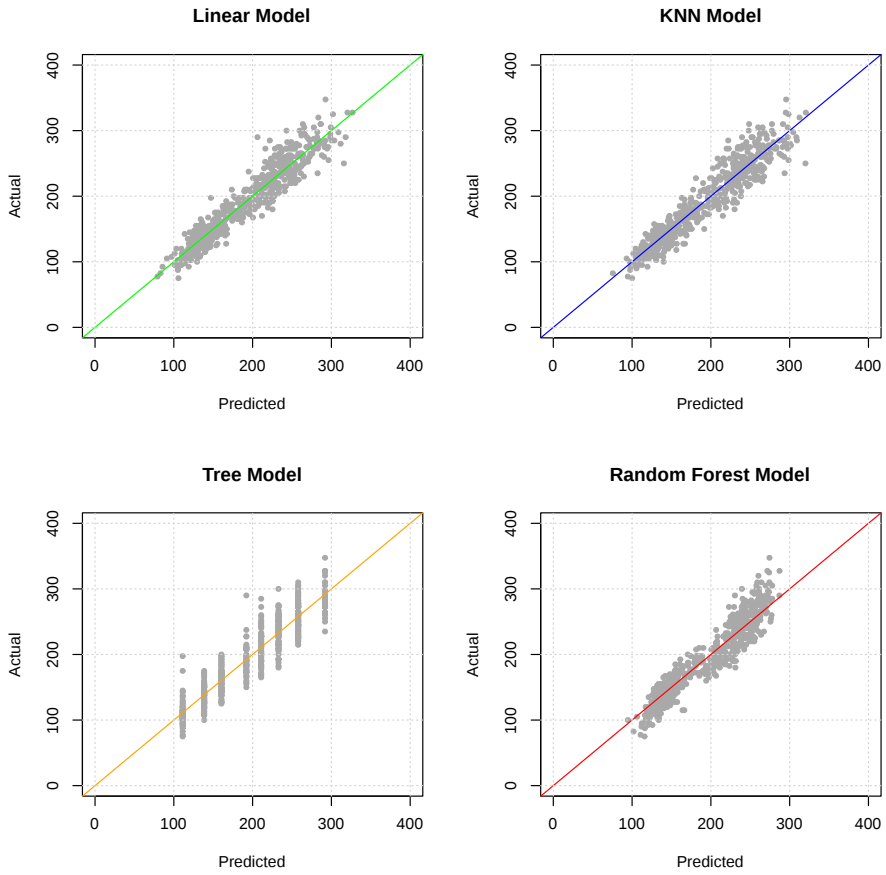


1.1.4 Modeling

```
dl_mod_form = formula(Deadlift ~ Sex + Bodyweight + Age + Squat + Bench)

set.seed(1)
lm_mod  = lm(dl_mod_form, data = pl_est)
knn_mod = caret::knnreg(dl_mod_form, data = pl_est)
rf_mod  = randomForest(dl_mod_form, data = pl_est)
rp_mod = rpart(dl_mod_form, data = pl_est)
```

- TODO: Note: we are not using `Name`. Why? We are not using `Total`. Why?
- TODO: look what happens with `Total`! You'll see it with `lm()`, you'll be optimistic with `randomForest()`.
- TODO: What variables are allowed? (With respect to real world problem.)
- TODO: What variables lead to the best predictions?

1.1.5 Model Evaluation




```
calc_rmse = function(actual, predicted) {
  sqrt(mean( (actual - predicted) ^ 2) )
}
```

```
c(calc_rmse(actual = pl_val$Deadlift, predicted = predict(lm_mod, pl_val)),
  calc_rmse(actual = pl_val$Deadlift, predicted = predict(knn_mod, pl_val)),
  calc_rmse(actual = pl_val$Deadlift, predicted = predict(rp_mod, pl_val)),
  calc_rmse(actual = pl_val$Deadlift, predicted = predict(rf_mod, pl_val)))
```

```
## [1] 18.26654 19.19625 21.68142 19.23643
```

```
reg_preds = map(list(lm_mod, knn_mod, rp_mod, rf_mod), predict, pl_val)
map_dbl(reg_preds, calc_rmse, actual = pl_val$Deadlift)
```

```
## [1] 18.26654 19.19625 21.68142 19.23643
```

- TODO: **Never** supply `data = df` to `predict()`. You have been warned.

```
knitr::include_graphics("img/sim-city.jpg")
```


```
calc_mae = function(actual, predicted) {
  mean(abs(actual - predicted))
}
```

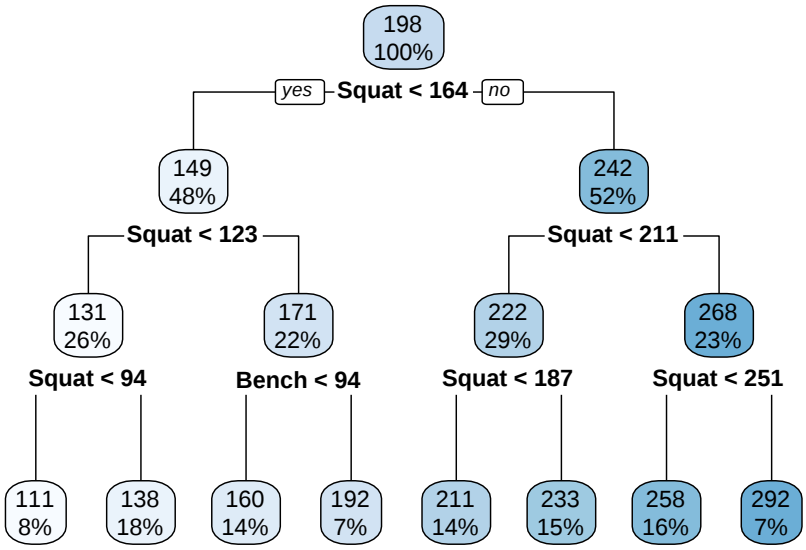
```
map_dbl(reg_preds, calc_mae,  actual = pl_val$Deadlift)
```

```
## [1] 14.38953 14.99748 17.14823 15.28626
```

```
reg_results = tibble(
  Model = c("Linear", "KNN", "Tree", "Forest"),
  RMSE = map_dbl(reg_preds, calc_rmse, actual = pl_val$Deadlift),
  MAE = map_dbl(reg_preds, calc_mae,  actual = pl_val$Deadlift))
```

Model	RMSE	MAE
Linear	18.26654	14.38953
KNN	19.19625	14.99748
Tree	21.68142	17.14823
Forest	19.23643	15.28626

1.1.6 Discussion




```
lm_mod_final = lm(dl_mod_form, data = pl_trn)
```

```
calc_rmse(actual = pl_tst$Deadlift,
          predicted = predict(lm_mod_final, pl_tst))
```

```
## [1] 22.29668
```

- TODO: Is this a good model?
- TODO: Is this model useful?

```
william_biscarri = tibble(
  Name = "William Biscarri",
  Age = 28,
  Sex = "M",
  Bodyweight = 83,
  Squat = 130,
  Bench = 90
)
```

```
predict(lm_mod_final, william_biscarri)
```

```
##       1
## 175.495
```

1.2 Classification: Handwritten Digits

1.2.1 Background

- TODO: https://en.wikipedia.org/wiki/MNIST_database
- TODO: http://yann.lecun.com/exdb/mnist/

1.2.2 Data

- TODO: How is this data pre-processed?
- TODO: https://gist.github.com/daviddalpiaz/ae62ae5ccd0bada4b9acd6dbc9008706
- TODO: https://github.com/itsrainingdata/mnistR
- TODO: https://pjreddie.com/projects/mnist-in-csv/
- TODO: http://varianceexplained.org/r/digit-eda/

```
mnist_trn = read_csv(file = "data/mnist_train_subest.csv")
mnist_tst = read_csv(file = "data/mnist_test.csv")
```

```
mnist_trn_y = as.factor(mnist_trn$X1)
mnist_tst_y = as.factor(mnist_tst$X1)

mnist_trn_x = mnist_trn[, -1]
mnist_tst_x = mnist_tst[, -1]
```

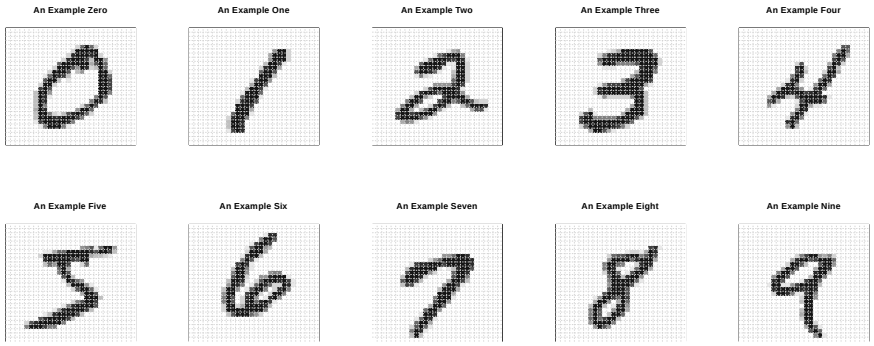
- TODO: If we were going to tune a model, we would need a validation split as well. We're going to be lazy and just fit a single random forest.
- TODO: This is an agreed upon split.

1.2.3 EDA

```
pixel_positions = expand.grid(j = sprintf("%02.0f", 1:28),
                              i = sprintf("%02.0f", 1:28))
pixel_names = paste("pixel", pixel_positions$i, pixel_positions$j, sep = "-")
```

```
colnames(mnist_trn_x) = pixel_names
colnames(mnist_tst_x) = pixel_names
```

```
show_digit = function(arr784, col = gray(12:1 / 12), ...) {
  image(matrix(as.matrix(arr784), nrow = 28)[, 28:1],
        col = col, xaxt = "n", yaxt = "n", ...)
  grid(nx = 28, ny = 28)
}
```



1.2.4 Modeling

```
set.seed(42)
mnist_rf = randomForest(x = mnist_trn_x, y = mnist_trn_y, ntree = 100)
```

1.2.5 Model Evaluation

```
mnist_tst_pred = predict(mnist_rf, mnist_tst_x)
mean(mnist_tst_pred == mnist_tst_y)
```

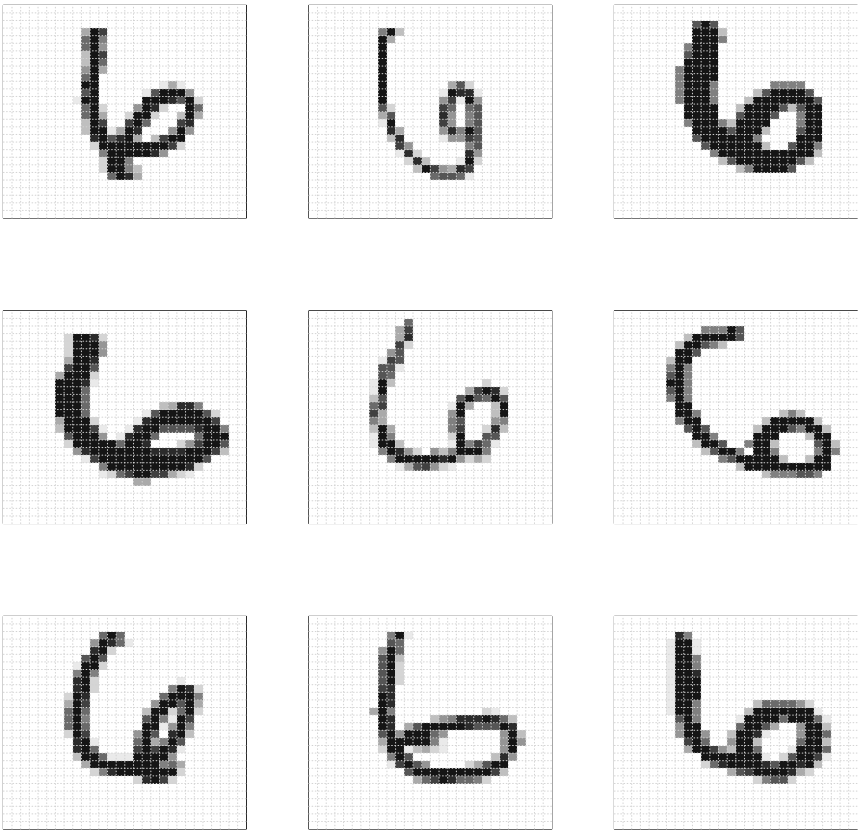
```
## [1] 0.8839
```

```
table(predicted = mnist_tst_pred, actual = mnist_tst_y)
```

```
##           actual
## predicted    0    1    2    3    4    5    6    7    8    9
##         0  959    0   14    6    1   15   22    1   10   10
##         1    0 1112    5    5    1   16    5    9    5    6
##         2    1    2  928   31    3    5   19   24   17    8
##         3    0    2   11  820    1   24    0    1   13   13
##         4    4    0   13    1  839   21   39   11   18   40
##         5    3    1    1   88    3  720   18    1   25    9
##         6    7    2   15    3   25   15  848    0   18    2
##         7    2    1   29   24    1   14    2  928   15   30
##         8    4   14   13   22    5   19    5    4  797    3
##         9    0    1    3   10  103   43    0   49   56  888
```

1.2.6 Discussion

```
par(mfrow = c(3, 3))
plot_mistake(actual = 6, predicted = 4)
```



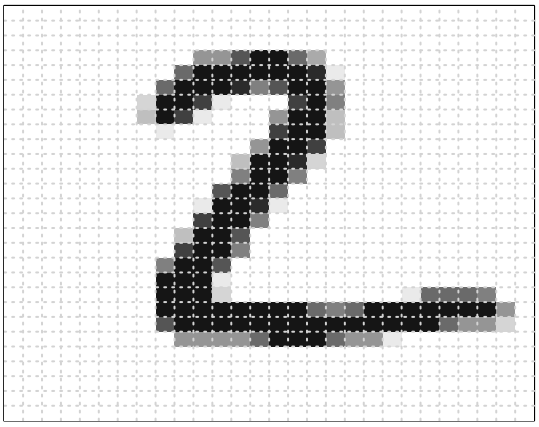
```
mnist_obs_to_check = 2
predict(mnist_rf, mnist_tst_x[mnist_obs_to_check, ], type = "prob")[1, ]
```

```
##    0    1    2    3    4    5    6    7    8    9
## 0.09 0.03 0.25 0.14 0.02 0.14 0.25 0.01 0.05 0.02
```

```
mnist_tst_y[mnist_obs_to_check]
```

```
## [1] 2
## Levels: 0 1 2 3 4 5 6 7 8 9
```

```
show_digit(mnist_tst_x[mnist_obs_to_check, ])
```



1.3 Clustering: NBA Players

1.3.1 Background

- https://www.youtube.com/watch?v=cuLprHh_BRg
- https://www.youtube.com/watch?v=1FBwSO_1Mb8
- https://www.basketball-reference.com/leagues/NBA_2019.html
- inspiration here, and others: http://blog.schochastics.net/post/analyzing-nba-player-data-ii-clustering/

1.3.2 Data

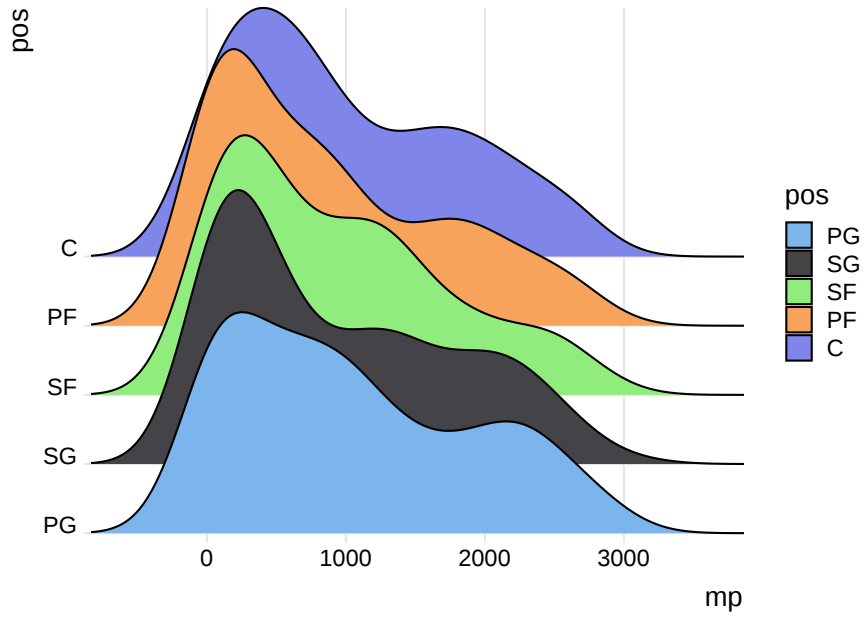
- https://www.basketball-reference.com/leagues/NBA_2019_totals.html

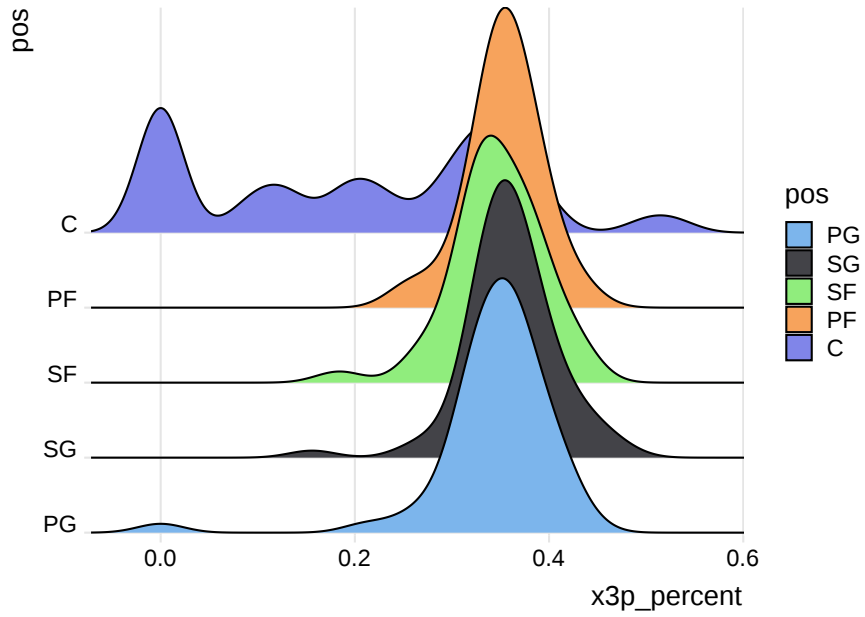
- https://www.basketball-reference.com/leagues/NBA_2019_per_minute.html
- https://www.basketball-reference.com/leagues/NBA_2019_per_poss.html
- https://www.basketball-reference.com/leagues/NBA_2019_advanced.html

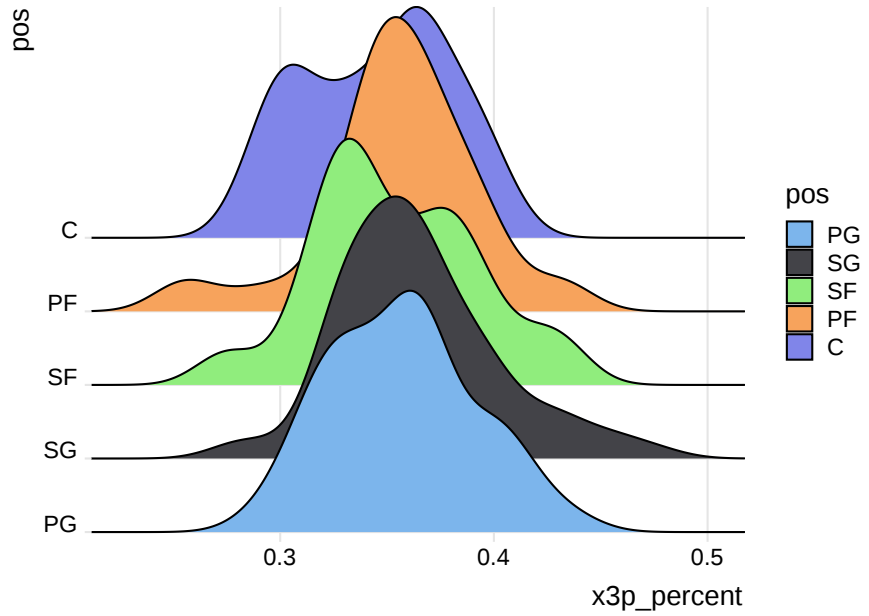
```
nba = scrape_nba_season_player_stats()
nba$pos = factor(nba$pos, levels = c("PG", "SG", "SF", "PF", "C"))
```

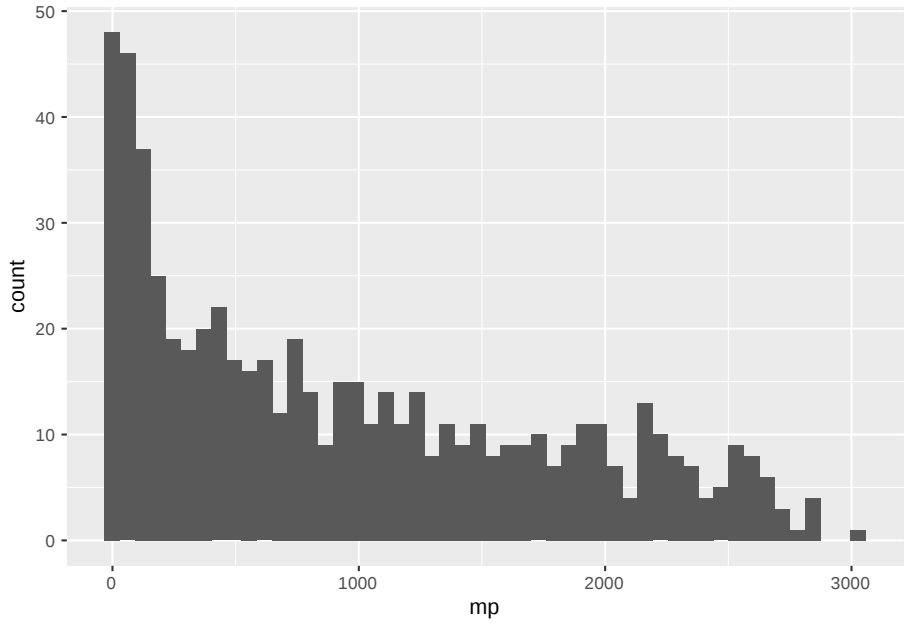
```
## # A tibble: 100 x 93
##    player_team pos     age tm        g    gs    mp    fg   fga fg_percent
##    <chr>       <fct> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>      <dbl>
##  1 Álex Abrin~ SG       25 OKC      31     2   588    56   157      0.357
##  2 Quincy Acy~ PF       28 PHO      10     0   123     4    18      0.222
##  3 Jaylen Ada~ PG       22 ATL      34     1   428    38   110      0.345
##  4 Steven Ada~ C        25 OKC      80    80  2669   481   809      0.595
##  5 Bam Adebay~ C        21 MIA      82    28  1913   280   486      0.576
##  6 Deng Adel ~ SF       21 CLE      19     3   194    11    36      0.306
##  7 DeVaughn A~ SG       25 DEN       7     0    22     3    10      0.3
##  8 LaMarcus A~ C        33 SAS      81    81  2687   684  1319      0.519
##  9 Rawle Alki~ SG       21 CHI      10     1   120    13    39      0.333
## 10 Grayson Al~ SG       23 UTA      38     2   416    67   178      0.376
## # ... with 90 more rows, and 83 more variables: x3p <dbl>, x3pa <dbl>,
## #   x3p_percent <dbl>, x2p <dbl>, x2pa <dbl>, x2p_percent <dbl>,
## #   e_fg_percent <dbl>, ft <dbl>, fta <dbl>, ft_percent <dbl>, orb <dbl>,
## #   drb <dbl>, trb <dbl>, ast <dbl>, stl <dbl>, blk <dbl>, tov <dbl>,
## #   pf <dbl>, pts <dbl>, fg_pm <dbl>, fga_pm <dbl>, fg_percent_pm <dbl>,
## #   x3p_pm <dbl>, x3pa_pm <dbl>, x3p_percent_pm <dbl>, x2p_pm <dbl>,
## #   x2pa_pm <dbl>, x2p_percent_pm <dbl>, ft_pm <dbl>, fta_pm <dbl>,
## #   ft_percent_pm <dbl>, orb_pm <dbl>, drb_pm <dbl>, trb_pm <dbl>,
## #   ast_pm <dbl>, stl_pm <dbl>, blk_pm <dbl>, tov_pm <dbl>, pf_pm <dbl>,
## #   pts_pm <dbl>, fg_pp <dbl>, fga_pp <dbl>, fg_percent_pp <dbl>,
## #   x3p_pp <dbl>, x3pa_pp <dbl>, x3p_percent_pp <dbl>, x2p_pp <dbl>,
## #   x2pa_pp <dbl>, x2p_percent_pp <dbl>, ft_pp <dbl>, fta_pp <dbl>,
## #   ft_percent_pp <dbl>, orb_pp <dbl>, drb_pp <dbl>, trb_pp <dbl>,
## #   ast_pp <dbl>, stl_pp <dbl>, blk_pp <dbl>, tov_pp <dbl>, pf_pp <dbl>,
## #   pts_pp <dbl>, o_rtg_pp <dbl>, d_rtg_pp <dbl>, per <dbl>,
## #   ts_percent <dbl>, x3p_ar <dbl>, f_tr <dbl>, orb_percent <dbl>,
## #   drb_percent <dbl>, trb_percent <dbl>, ast_percent <dbl>,
## #   stl_percent <dbl>, blk_percent <dbl>, tov_percent <dbl>,
## #   usg_percent <dbl>, ows <dbl>, dws <dbl>, ws <dbl>, ws_48 <dbl>,
## #   obpm <dbl>, dbpm <dbl>, bpm <dbl>, vorp <dbl>
```

1.3.3 EDA


pos
C
PF
SF
SG
PG
0
1000
2000
3000
mp
pos
PG
SG
SF
PF
C


pos
C
PF
SF
SG
PG
0.0
0.2
0.4
0.6
x3p_percent
pos
PG
SG
SF
PF
C

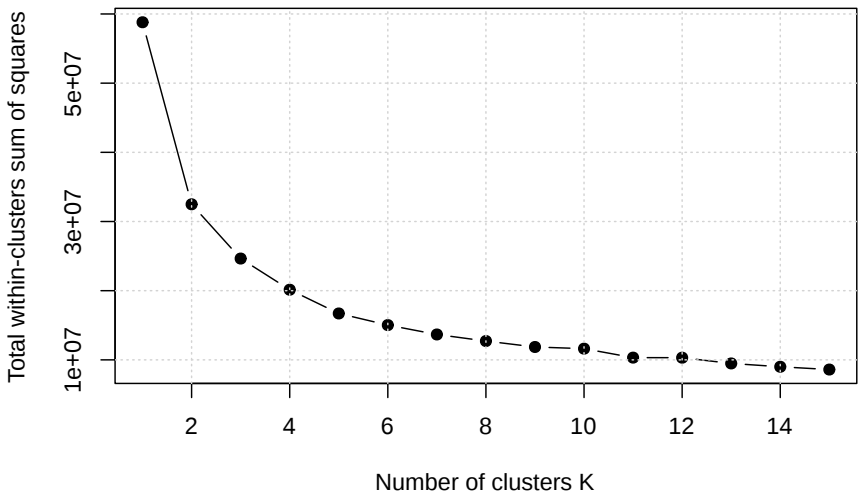





```
nba_for_clustering = nba %>%
  filter(mp > 2000) %>%
  column_to_rownames("player_team") %>%
  select(-pos, -tm)
```

1.3.4 Modeling

```
set.seed(42)

# function to compute total within-cluster sum of square
wss = function(k, data) {
  kmeans(x = data, centers = k, nstart = 10)$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k_values = 1:15

# extract wss for 2-15 clusters
wss_values = map_dbl(k_values, wss, data = nba_for_clustering)

plot(k_values, wss_values,
       type = "b", pch = 19, frame = TRUE,
       xlab = "Number of clusters K",
       ylab = "Total within-clusters sum of squares")
grid()
```

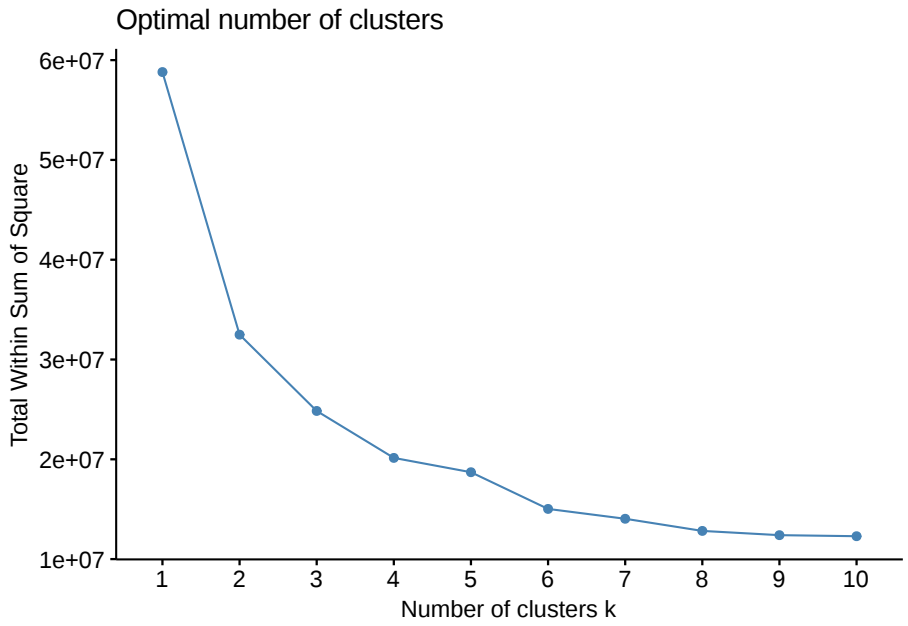


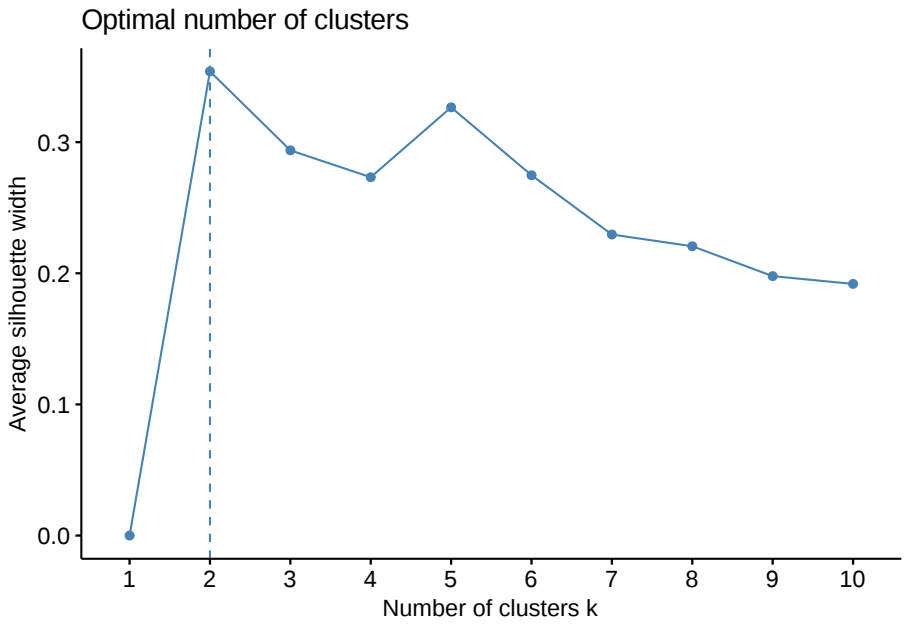
- TODO: K-Means likes clusters of roughly equal size.
- TODO: http://varianceexplained.org/r/kmeans-free-lunch/

```
nba_hc = hclust(dist(nba_for_clustering))
nba_hc_clust = cutree(nba_hc, k = 5)
table(nba_hc_clust)
```

```
## nba_hc_clust
##  1  2  3  4  5
## 38 13 28 11  1
```

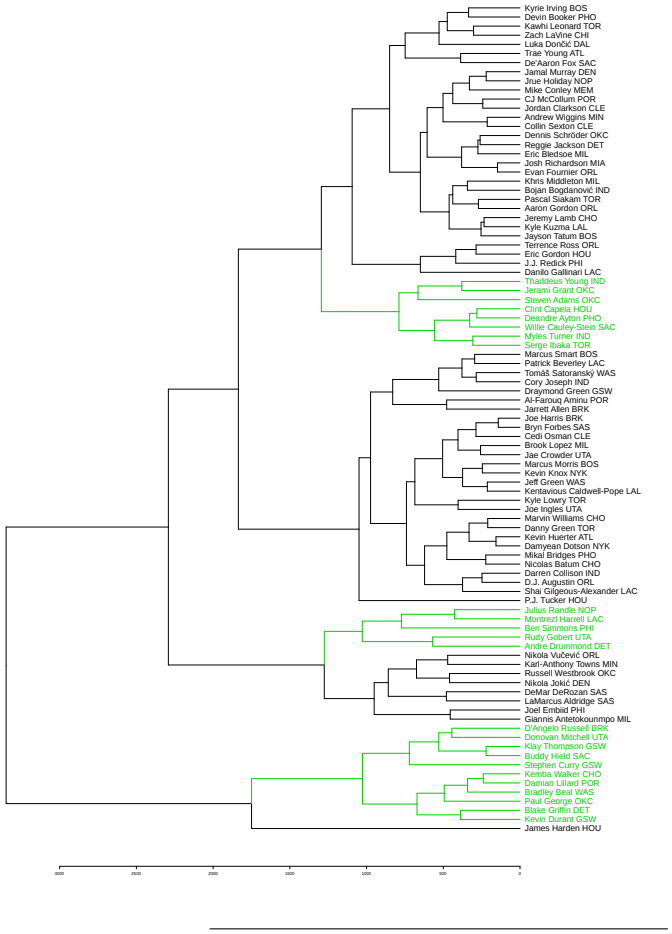
1.3.5 Model Evaluation




Optimal number of clusters
Average silhouette width
0.0
0.1
0.2
0.3
1
2
3
4
5
6
7
8
9
10
Number of clusters k


Cluster plot
Rudy Gobert UTA
Clint Capela HOU
Jarrett Allen BRK
Andre Drummond DET
Deandre Ayton PHO
Willie Cauley-Stein SAC
Myles Turner IND
Draymond Green GSW
P.J. Tucker HOU
Serge Ibaka TOR
Montrezl Harrell LAC
Brook Lopez MIL
Nicolas Batum CHO
Danny Green TOR
Tomás Satoransky WAS
Kevin Huerter ATL
Marcus Smart BOS
Damyean Dotson NYK
Ben Simmons PHI
Pascal Siakam TOR
Shai Gilgeous-Alexander OKC
Marcus Morris BOS
Aaron Gordon ORL
Joe Ingles UTA
Bam Adebayo MIA
Jayson Tatum BOS
Derrick White SAS
Barren Collison IND
LaMarcus Aldridge SAS
Nikola Vučević ORL
Kyle Kuzma LAL
D.J. Augustin ORL
Julius Randle NOP
Jeremy Lamb CHI
Eric Bledsoe MIL
Kyle Lowry TOR
Eric Gordon HOU
Karl-Anthony Towns MIN
Nikola Jokić DEN
Bojan Bogdanović IND
Joel Embiid PHI
Dennis Schröder OKC
Collin Sexton CLE
DeMar DeRozan SAS
CJ McCollum POR
Klay Thompson GSW
Buddy Hield SAC
Russell Westbrook OKC
Zach LaVine CHI
Kevin Durant GSW
Blake Griffin DET
Mike Conley MEM
Bradley Beal WAS
Donovan Mitchell UTA
Trae Young ATL
Devin Booker PHO
Paul George OKC
D'Angelo Russell BRK
Stephen Curry GSW
Damian Lillard POR
Kemba Walker CHO
James Harden HOU
cluster
1
2
3
4
5
Dim2 (23.2%)
Dim1 (35.9%)
10
5
0
-5
-10
-15
-10
0

1.3.6 Discussion



Chapter 2

Computing

This is not a book about R. It is however, a book that uses R. Because of this, you will need to be familiar with R. The text will point out some thing about R along the way, but some previous study of R is necessary.

The following (freely available) readings are highly recommended:

- Hands-On Programming with R - *Garrett Grolemund*
  - If you have never used R or RStudio before, Part 1, Chapters 1 - 3, will be useful.
- R for Data Science - *Garrett Grolemund, Hadley Wickham*
  - This book helps getting you up to speed working with data in R. While it is a lot of reading, Chapters 1 - 21 are highly recommended.
- Advanced R - *Hadley Wickham*
  - Part I, Chapters 1 - 8, of this book will help create a mental model for working with R. These chapters are not an easy read, so they should be returned to often. (Chapter 2 could be safely skipped for our purposes, but is important if you will use R in the long term.)

If you are a UIUC student who took the course STAT 420, the first six chapters of that book could serve as a nice refresher.

- Applied Statistics with R - *David Dalpiaz*

2.1 Resources

The following resources are more specific or more advanced, but could still prove to be useful.

2.1.1 R

- Efficient R programming
- R Programming for Data Science
- R Graphics Cookbook
- Modern Dive
- The `tidyverse` Website
  - `dplyr` Website
  - `readr` Website
  - `tibble` Website
  - `forcats` Website

2.1.2 RStudio

- RStudio IDE Cheatsheet
- RStudio Resources

2.1.3 R Markdown

- R Markdown Cheatsheet
- R Markdown: The Definitive Guide - *Yihui Xie, J. J. Allaire, Garrett Grolemund*
- R4DS R Markdown Chapter

2.1.3.1 Markdown

- Daring Fireball - Markdown: Basics
- GitHub - Mastering Markdown
- CommonMark

2.2 BSL Idioms

Things here supersede everything above.

2.2.1 Reference Style

- `tidyverse` Style Guide

2.2.2 BSL Style Overrides

- TODO: = instead of <-
  - http://thecoatlessprofessor.com/programming/an-opinionated-tale-of-why-you-should-replace---with-/
- TODO: never use T or F, only TRUE or FALSE

```
FALSE == TRUE
```

```
## [1] FALSE
```

```
F     == TRUE
```

```
## [1] FALSE
```

```
F     =  TRUE
F     == TRUE
```

```
## [1] TRUE
```

- TODO: never ever ever use attach()
- TODO: never ever ever use <<-
- TODO: never ever ever use setwd() or set a working directory some other way
- TODO: a newline before and after any chunk
- TODO: use headers appropriately! (short names, good structure)
- TODO: never ever ever put spaces in filenames. use -. (others will use _.)
- TODO: load all needed packages at the beginning of an analysis in a single chunk (TODO: pros and cons of this approach)
- TODO: one plot per chunk! no other printed output

Be consistent...

- with yourself!
- with your group!
- with your organization!

```
set.seed(1337);mu=10;sample_size=50;samples=100000;
x_bars=rep(0, samples)
for(i in 1:samples)
{
x_bars[i]=mean(rpois(sample_size,lambda = mu))}
x_bar_hist=hist(x_bars,breaks=50,main="Histogram of Sample Means",xlab="Sample Means",col="darkor
mean(x_bars>mu-2*sqrt(mu)/sqrt(sample_size)&x_bars<mu+2*sqrt(mu)/sqrt(sample_size))
```

2.2.3 Objects and Functions

To understand computations in R, two slogans are helpful:

- Everything that exists is an object.
- Everything that happens is a function call.

— John Chambers

- TODO: Functions + Objects
  - these are the inputs and outputs of functions:
    - functions
    - vectors
    - lists
    - tibbles (dfs)

2.2.4 Print versus Return

```
cars_mod = lm(dist ~ speed, data = cars)
```

```
summary(cars_mod)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

```
is.list(summary(cars_mod))
```

```
## [1] TRUE
```

```
names(summary(cars_mod))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```
str(summary(cars_mod))
```

```
## List of 11
```

```
##  $ call         : language lm(formula = dist ~ speed, data = cars)
##  $ terms        :Classes 'terms', 'formula'  language dist ~ speed
##   .. ..- attr(*, "variables")= language list(dist, speed)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "dist" "speed"
##   .. .. .. ..$ : chr "speed"
##   .. ..- attr(*, "term.labels")= chr "speed"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(dist, speed)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "dist" "speed"
##  $ residuals    : Named num [1:50] 3.85 11.85 -5.95 12.05 2.12 ...
##   ..- attr(*, "names")= chr [1:50] "1" "2" "3" "4" ...
##  $ coefficients : num [1:2, 1:4] -17.579 3.932 6.758 0.416 -2.601 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "(Intercept)" "speed"
##   .. ..$ : chr [1:4] "Estimate" "Std. Error" "t value" "Pr(>|t|)"
##  $ aliased      : Named logi [1:2] FALSE FALSE
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "speed"
##  $ sigma        : num 15.4
##  $ df           : int [1:3] 2 48 2
##  $ r.squared    : num 0.651
##  $ adj.r.squared: num 0.644
##  $ fstatistic   : Named num [1:3] 89.6 1 48
##   ..- attr(*, "names")= chr [1:3] "value" "numdf" "dendf"
##  $ cov.unscaled : num [1:2, 1:2] 0.19311 -0.01124 -0.01124 0.00073
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "(Intercept)" "speed"
##   .. ..$ : chr [1:2] "(Intercept)" "speed"
##  - attr(*, "class")= chr "summary.lm"
```

```
# RStudio only
View(summary(cars_mod))
```

2.2.5 Help

- TODO: ?, google, stack overflow, (office hours, course forums)

2.2.6 Keyboard Shortcuts

- TODO: copy-paste, switch program, switch tab, etc…
- TODO: TAB!!!
- TODO: new chunk!
- TODO: style!
- TODO: keyboard shortcut for keyboard shortcut

2.3 Common Issues

- TODO: cannot find function called ””

- TODO: https://stat545.com/
- TODO: https://atrebas.github.io/post/2019-01-15-2018-learning/

Chapter 3

Estimation

- TODO: Where we are going, estimating conditional means and distributions.
- TODO: estimation = learning. "learning from data." what are we learning about? often parameters.
- TODO: http://stat400.org
- TODO: http://stat420.org

3.1 Probability

- TODO: See Appendix A
- TODO: In R, `d*()`, `p*()`, `q*()`, `r*()`

3.2 Statistics

- TODO: parameters are a function of the population distribution
- TODO: statistics are a function of data.
- TODO: parameters:population::statistics::data
- TODO: statistic vs value of a statistic

3.3 Estimators

- TODO: estimator vs estimate
- TODO: Why such a foucs on the mean, E[X]? Because E[(X - a)^2] is minimized by E[X]
  - https://www.benkuhn.net/squared

– https://news.ycombinator.com/item?id=9556459

3.3.1 Properties

3.3.1.1 Bias

$$\text{bias}\left[\hat{\theta}\right] \triangleq \mathbb{E}\left[\hat{\theta}\right] - \theta$$

3.3.1.2 Variance

$$\text{var}\left[\hat{\theta}\right] \triangleq \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right]\right)^2\right]$$

3.3.1.3 Mean Squared Error

$$\text{MSE}\left[\hat{\theta}\right] \triangleq \mathbb{E}\left[\left(\hat{\theta} - \theta\right)^2\right] = \text{var}\left[\hat{\theta}\right] + \left(\text{Bias}\left[\hat{\theta}\right]\right)^2$$

3.3.1.4 Consistency

An estimator $\hat{\theta}_n$ is said to be a **consistent estimator** of θ if, for any positive ϵ,

$$\lim_{n\to\infty} P\left(\left|\hat{\theta}_n - \theta\right| \leq \epsilon\right) = 1$$

or, equivalently,

$$\lim_{n\to\infty} P\left(\left|\hat{\theta}_n - \theta\right| > \epsilon\right) = 0$$

We say that $\hat{\theta}_n$ **converges in probability** to θ and we write $\hat{\theta}_n \xrightarrow{P} \theta$.

3.3.2 Methods

- TODO: MLE

Given a random sample $X_1, X_2, \ldots, X_n$ from a population with parameter θ and density or mass $f(x \mid \theta)$, we have:

The Likelihood, $L(\theta)$,

$$L(\theta) = f(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} f(x_i \mid \theta)$$

The **Maximum Likelihood Estimator**, $\hat{\theta}$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ L(\theta) = \underset{\theta}{\operatorname{argmax}}\ \log L(\theta)$$

- TODO: Invariance Principle

If $\hat{\theta}$ is the MLE of θ and the function $h(\theta)$ is continuous, then $h(\hat{\theta})$ is the MLE of $h(\theta)$.

- TODO: MOM
- TODO: https://daviddalpiaz.github.io/stat3202-sp19/notes/fitting.html
- TODO: ECDF: https://en.wikipedia.org/wiki/Empirical_distribution_function

Chapter 4

Regression

BLUF: Use **regression**, which is one of the two **supervised learning** tasks (the other being **classification**) to make predictions of new observations of **numeric *response* variables.** Start by randomly splitting the data (which includes both the response and the **features**) into a **test set** and a **training set**. Do not use the test data for anything other than supplying a final assesment of how well a chosen model performs at the prediction task. That is, never use the test data to make *any* modeling decisions. Use the training data however you please, but it is recommended to further split this data into an **estimation set** and a **validation set**. The estimation set should be used to **train** models for evaluation. For example, use the estimation data to learn the **model parameters** of a **parametric model**. Do not use data used in training of models (the estimation data) when evaluating models as doing so will mask **overfitting** of **complex** (flexible) models. Use the `lm()` function to train **linear models**. Use the `knnreg()` function from the `caret` pacakge to train **k-nearest neighbors models**. Use the `rpart()` function from the `rpart` package to train **decision tree models**. Use the validation set to evaluate models that have been trained using the estimation data. For example, use the validation data to select the value of **tuning parameters** that are often used in **non-parametric models**. Use numeric metrics such as **root-mean-square error (RMSE)** or graphical summaries such as **actual versus predicted plots**. Although it ignores some practical and statistical considerations (which will be discussed later), the model that acheives the lowest RMSE on the validation data will be deemed the "best" model. After finding this model, refit the model to the entire training dataset. Report the RMSE of this model on the test data as a final quantification of performance.

- TODO: add ISL readings
- TODO: <www.stat420.org>
- TODO: add "why least squares?" readings

```
library(tibble)
library(caret)
library(rpart)
library(knitr)
library(kableExtra)
library(dplyr)
```

4.1 Setup

$$Y = f(X) + \epsilon$$

- TODO: signal $f(X)$
- TODO: noise ϵ
- TODO: goal: learn the signal, not the noise
- TODO: random variables versus potential realized values

$$X = (X_1, X_2, \ldots, X_p)$$

$$x = (x_1, x_2, \ldots, x_p)$$

$$\mathbb{E}\left[\left(Y - f(X)^2\right)\right] =$$

- TODO: define regression function

$$f(x) = \mathbb{E}[Y \mid X = x]$$

- TODO: want to learn these "things" which are regression functions

```
line_reg_fun = function(x) {
  x
}
```

```
quad_reg_fun = function(x) {
  x ^ 2
}
```

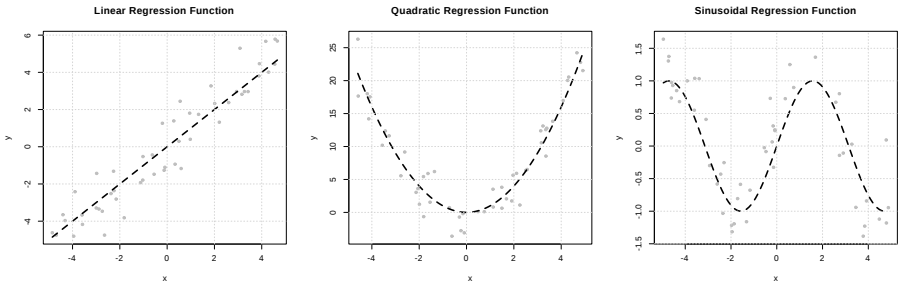
```
sine_reg_fun = function(x) {
  sin(x)
}
```

```
gen_sim_data = function(f, sample_size = 50, sd = 1) {
  x = runif(n = sample_size, min = -5, max = 5)
  y = rnorm(n = sample_size, mean = f(x), sd = sd)
```

```
  tibble::tibble(x = x, y = y)
}
```

```
set.seed(5)
line_data = gen_sim_data(f = line_reg_fun, sample_size = 50, sd = 1.0)
quad_data = gen_sim_data(f = quad_reg_fun, sample_size = 50, sd = 2.0)
sine_data = gen_sim_data(f = sine_reg_fun, sample_size = 50, sd = 0.5)
```

```
set.seed(42)
line_data_unseen = gen_sim_data(f = line_reg_fun, sample_size = 100000, sd = 1.0)
quad_data_unseen = gen_sim_data(f = quad_reg_fun, sample_size = 100000, sd = 2.0)
sine_data_unseen = gen_sim_data(f = sine_reg_fun, sample_size = 100000, sd = 0.5)
```



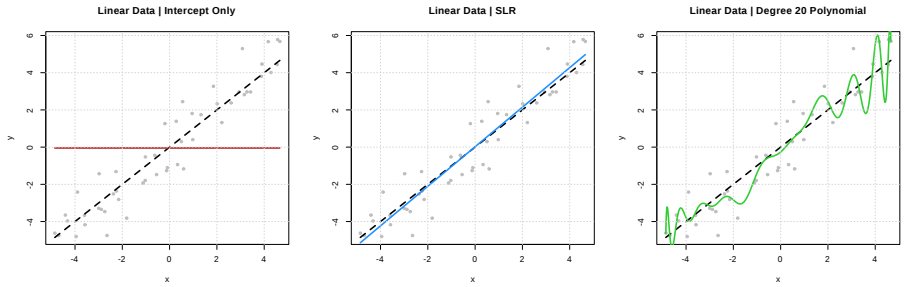
4.2 Modeling

- TODO: for now, only use formula syntax
  - https://rviews.rstudio.com/2017/02/01/the-r-formula-method-the-good-parts/
  - https://rviews.rstudio.com/2017/03/01/the-r-formula-method-the-bad-parts/

4.2.1 Linear Models

- TODO: assume form of mean relationship. linear combination
- TODO: how to go from y = b0 + b1x1 + … + eps to `lm(y ~ stuff)`
- TODO: least squares, least squares is least squares (difference in assumptions)

```
lm_line_int = lm(y ~ 1,                    data = line_data)
lm_line_slr = lm(y ~ poly(x, degree = 1),  data = line_data)
lm_line_ply = lm(y ~ poly(x, degree = 20), data = line_data)
```

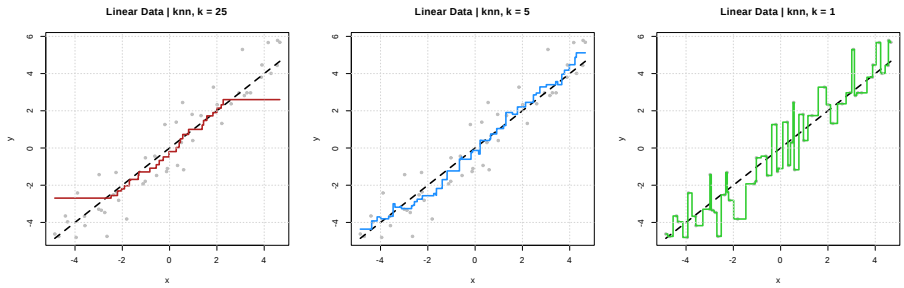


4.2.2 k-Nearest Neighbors

- TODO: `caret::knnreg()`
- TODO: for now, don't worry about scaling, factors, etc.

4.2.2.1 Linear Data

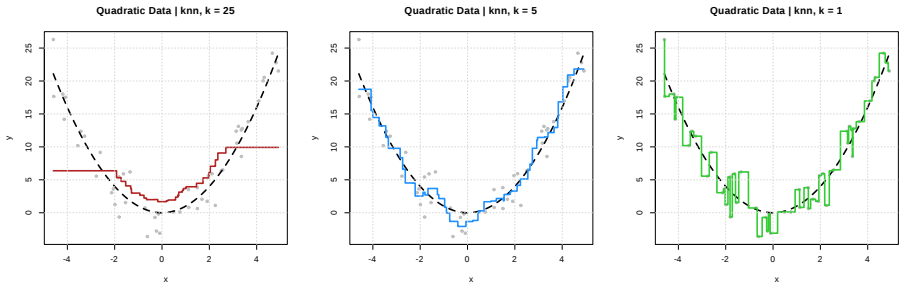
```
knn_line_25 = knnreg(y ~ x, data = line_data, k = 25)
knn_line_05 = knnreg(y ~ x, data = line_data, k = 5)
knn_line_01 = knnreg(y ~ x, data = line_data, k = 1)
```



k	Train RMSE	Test RMSE
25	1.406	1.379
5	0.931	1.061
1	0.000	1.409

4.2.2.2 Quadratic Data

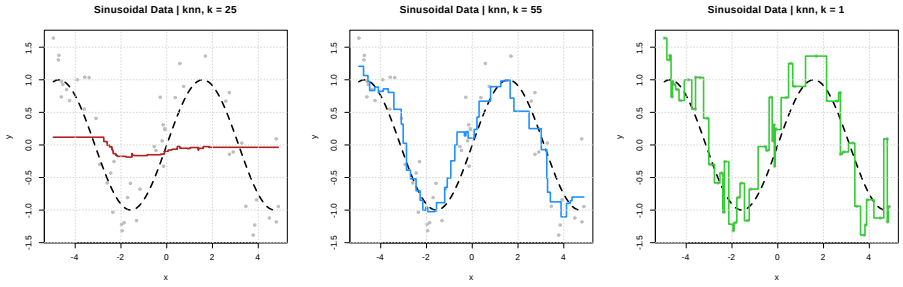
```
knn_quad_25 = knnreg(y ~ x, data = quad_data, k = 25)
knn_quad_05 = knnreg(y ~ x, data = quad_data, k = 5)
knn_quad_01 = knnreg(y ~ x, data = quad_data, k = 1)
```



k	Train RMSE	Test RMSE
25	6.393	6.564
5	2.236	2.509
1	0.000	3.067

4.2.2.3 Sinusoidal Data

```
knn_sine_25 = knnreg(y ~ x, data = sine_data, k = 25)
knn_sine_05 = knnreg(y ~ x, data = sine_data, k = 5)
knn_sine_01 = knnreg(y ~ x, data = sine_data, k = 1)
```



k	Train RMSE	Test RMSE
25	0.814	0.841
5	0.349	0.570
1	0.000	0.647

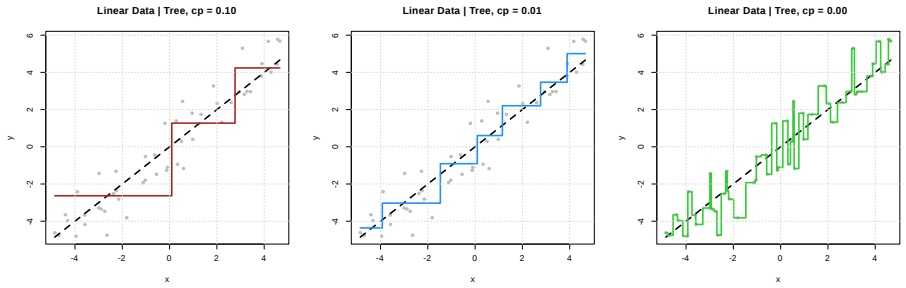
4.2.3 Decision Trees

- TODO: `rpart::rpart()`
- TODO: https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf
- TODO: http://www.milbo.org/doc/prp.pdf

- TODO: maybe notes about pruning and CV

4.2.3.1 Linear Data

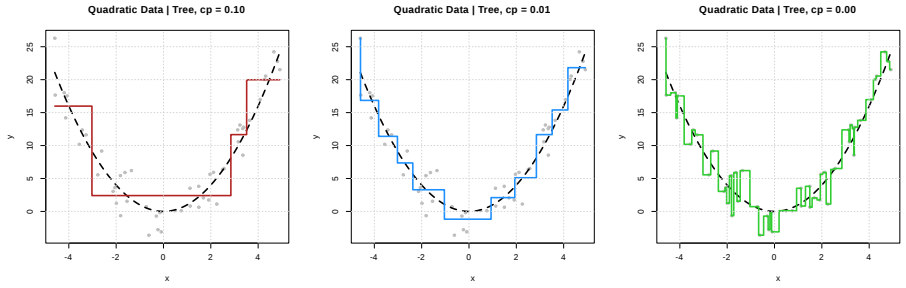
```
tree_line_010 = rpart(y ~ x, data = line_data, cp = 0.10, minsplit = 2)
tree_line_001 = rpart(y ~ x, data = line_data, cp = 0.01, minsplit = 2)
tree_line_000 = rpart(y ~ x, data = line_data, cp = 0.00, minsplit = 2)
```



k	Train RMSE	Test RMSE
25	1.394	1.548
5	0.914	1.144
1	0.000	1.409

4.2.3.2 Quadratic Data

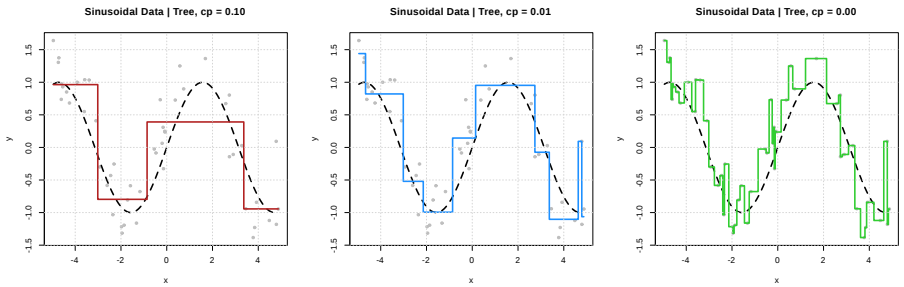
```
tree_quad_010 = rpart(y ~ x, data = quad_data, cp = 0.10, minsplit = 2)
tree_quad_001 = rpart(y ~ x, data = quad_data, cp = 0.01, minsplit = 2)
tree_quad_000 = rpart(y ~ x, data = quad_data, cp = 0.00, minsplit = 2)
```



k	Train RMSE	Test RMSE
25	3.376	3.869
5	1.692	2.621
1	0.000	3.067

4.2.3.3 Sinusoidal Data

```
tree_sine_010 = rpart(y ~ x, data = sine_data, cp = 0.10, minsplit = 2)
tree_sine_001 = rpart(y ~ x, data = sine_data, cp = 0.01, minsplit = 2)
tree_sine_000 = rpart(y ~ x, data = sine_data, cp = 0.00, minsplit = 2)
```



k	Train RMSE	Test RMSE
25	0.414	0.659
5	0.235	0.629
1	0.000	0.647

4.3 Procedure

- TODO: Look at data
- TODO: Pick candiate models
- TODO: Tune / train models
- TODO: Pick "best" model
  - based on validation RMSE (note the issues with this)
- TODO: Use best model / report test metrics

4.4 Data Splitting

- TODO: want to generalize to unseen data
- TODO: for now, all variables should either be numeric, or factor

- TODO: Training (Train) Data
- TODO: Testing (Test) Data
- TODO: Estimation Data
- TODO: Validation Data
  - https://en.wikipedia.org/wiki/Infinite_monkey_theorem

$$\mathcal{D} = \{(x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, \ i = 1, 2, \ldots n\}$$

$$\mathcal{D} = \mathcal{D}_{\texttt{trn}} \cup \mathcal{D}_{\texttt{tst}}$$

$$\mathcal{D}_{\texttt{trn}} = \mathcal{D}_{\texttt{est}} \cup \mathcal{D}_{\texttt{val}}$$

4.5 Metrics

- TODO: RMSE

$$\text{rmse}\left(\hat{f}_{\texttt{set}}, \mathcal{D}_{\texttt{set}}\right) = \sqrt{\frac{1}{n_{\texttt{set}}} \sum_{i \in \texttt{set}} \left(y_i - \hat{f}_{\texttt{set}}(x_i)\right)^2}$$

$$\text{RMSE}_{\texttt{trn}} = \text{rmse}\left(\hat{f}_{\texttt{est}}, \mathcal{D}_{\texttt{est}}\right) = \sqrt{\frac{1}{n_{\texttt{est}}} \sum_{i \in \texttt{est}} \left(y_i - \hat{f}_{\texttt{est}}(x_i)\right)^2}$$

$$\text{RMSE}_{\texttt{val}} = \text{rmse}\left(\hat{f}_{\texttt{est}}, \mathcal{D}_{\texttt{val}}\right) = \sqrt{\frac{1}{n_{\texttt{val}}} \sum_{i \in \texttt{val}} \left(y_i - \hat{f}_{\texttt{est}}(x_i)\right)^2}$$

$$\text{RMSE}_{\texttt{tst}} = \text{rmse}\left(\hat{f}_{\texttt{trn}}, \mathcal{D}_{\texttt{tst}}\right) = \sqrt{\frac{1}{n_{\texttt{tst}}} \sum_{i \in \texttt{tst}} \left(y_i - \hat{f}_{\texttt{trn}}(x_i)\right)^2}$$

- TODO: MAE
- TODO: MAPE
  - https://en.wikipedia.org/wiki/Mean_absolute_percentage_error
  - but probably don't use

```
calc_rmse = function(model, data, response) {
  actual = data[[response]]
  predicted = predict(model, data)
  sqrt(mean((actual - predicted) ^ 2))
}
```

4.6 Model Complexity

- TODO: what determines the complexity of the above models?
  - lm: terms, xforms, interactions
  - knn: k (also terms, xforms, interactions)
  - tree: cp (with rpart, also others that we'll keep mostly hidden) (also terms, xforms, interactions)

4.7 Overfitting

- TODO: too complex
- TODO: usual picture with training and validation error
- TODO: define for the purposes of this course

4.8 Multiple Features

- TODO: more features = more complex
- TODO: how do the three models add additional features?

4.9 Example Analysis

- TODO: Diamonds analysis
- TODO: model.matrix()

4.10 MISC TODOS

- lex fridman with ian: dataset (represent), model, optimize
  - https://www.youtube.com/watch?v=Z6rxFNMGdn0
- want to minimize E[(y - y_hat)^2]
- predict() creates estimate of E[Y|X] with supplied model

Chapter 5

Bias–Variance Tradeoff

Consider the general regression setup where we are given a random pair $(X, Y) \in \mathbb{R}^p \times \mathbb{R}$. We would like to "predict" Y with some function of X, say, $f(X)$.

To clarify what we mean by "predict," we specify that we would like $f(X)$ to be "close" to Y. To further clarify what we mean by "close," we define the **squared error loss** of estimating Y using $f(X)$.

$$L(Y, f(X)) \triangleq (Y - f(X))^2$$

Now we can clarify the goal of regression, which is to minimize the above loss, on average. We call this the **risk** of estimating Y using $f(X)$.

$$R(Y, f(X)) \triangleq \mathbb{E}[L(Y, f(X))] = \mathbb{E}_{X,Y}[(Y - f(X))^2]$$

Before attempting to minimize the risk, we first re-write the risk after conditioning on X.

$$\mathbb{E}_{X,Y}\left[(Y - f(X))^2\right] = \mathbb{E}_X \mathbb{E}_{Y|X}\left[(Y - f(X))^2 \mid X = x\right]$$

Minimizing the right-hand side is much easier, as it simply amounts to minimizing the inner expectation with respect to $Y \mid X$, essentially minimizing the risk pointwise, for each x.

It turns out, that the risk is minimized by the conditional mean of Y given X,

$$f(x) = \mathbb{E}(Y \mid X = x)$$

which we call the **regression function**.

Note that the choice of squared error loss is somewhat arbitrary. Suppose instead we chose absolute error loss.

$$L(Y, f(X)) \triangleq |Y - f(X)|$$

The risk would then be minimized by the conditional median.

$$f(x) = \text{median}(Y \mid X = x)$$

Despite this possibility, our preference will still be for squared error loss. The reasons for this are numerous, including: historical, ease of optimization, and protecting against large deviations.

Now, given data $\mathcal{D} = (x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}$, our goal becomes finding some $\hat{f}$ that is a good estimate of the regression function f. We'll see that this amounts to minimizing what we call the reducible error.

5.1 Reducible and Irreducible Error

Suppose that we obtain some $\hat{f}$, how well does it estimate f? We define the **expected prediction error** of predicting Y using $\hat{f}(X)$. A good $\hat{f}$ will have a low expected prediction error.

$$\text{EPE}\left(Y, \hat{f}(X)\right) \triangleq \mathbb{E}_{X,Y,\mathcal{D}}\left[\left(Y - \hat{f}(X)\right)^2\right]$$

This expectation is over X, Y, and also $\mathcal{D}$. The estimate $\hat{f}$ is actually random depending on the sampled data $\mathcal{D}$. We could actually write $\hat{f}(X, \mathcal{D})$ to make this dependence explicit, but our notation will become cumbersome enough as it is.

Like before, we'll condition on X. This results in the expected prediction error of predicting Y using $\hat{f}(X)$ when $X = x$.

$$\text{EPE}\left(Y, \hat{f}(x)\right) = \mathbb{E}_{Y|X,\mathcal{D}}\left[\left(Y - \hat{f}(X)\right)^2 \mid X = x\right] = \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(f(x) - \hat{f}(x)\right)^2\right]}_{\text{reducible error}} + \underbrace{\mathbb{V}_{Y|X}\left[Y \mid X = x\right]}_{\text{irreducible error}}$$

A number of things to note here:

- The expected prediction error is for a random Y given a fixed x and a random $\hat{f}$. As such, the expectation is over $Y \mid X$ and $\mathcal{D}$. Our estimated function $\hat{f}$ is random depending on the sampled data, $\mathcal{D}$, which is used to perform the estimation.
- The expected prediction error of predicting Y using $\hat{f}(X)$ when $X = x$ has been decomposed into two errors:
  - The **reducible error**, which is the expected squared error loss of estimation $f(x)$ using $\hat{f}(x)$ at a fixed point x. The only thing that is random here is $\mathcal{D}$, the data used to obtain $\hat{f}$. (Both f and x are fixed.) We'll often call this reducible error the **mean squared error** of estimating $f(x)$ using $\hat{f}$ at a fixed point x.

$$\text{MSE}\left(f(x), \hat{f}(x)\right) \triangleq \mathbb{E}_{\mathcal{D}}\left[\left(f(x) - \hat{f}(x)\right)^2\right]$$

  - The **irreducible error**. This is simply the variance of Y given that $X = x$, essentially noise that we do not want to learn. This is also called the **Bayes error**.

As the name suggests, the reducible error is the error that we have some control over. But how do we control this error?

5.2 Bias-Variance Decomposition

After decomposing the expected prediction error into reducible and irreducible error, we can further decompose the reducible error.

Recall the definition of the **bias** of an estimator.

$$\text{bias}(\hat{\theta}) \triangleq \mathbb{E}\left[\hat{\theta}\right] - \theta$$

Also recall the definition of the **variance** of an estimator.

$$\mathbb{V}(\hat{\theta}) = \text{var}(\hat{\theta}) \triangleq \mathbb{E}\left[(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right])^2\right]$$

Using this, we further decompose the reducible error (mean squared error) into bias squared and variance.

$$\text{MSE}\left(f(x), \hat{f}(x)\right) = \mathbb{E}_{\mathcal{D}}\left[\left(f(x) - \hat{f}(x)\right)^2\right] = \underbrace{\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2}_{\text{bias}^2\left(\hat{f}(x)\right)} + \underbrace{\mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2\right]}_{\text{var}\left(\hat{f}(x)\right)}$$

This is actually a common fact in estimation theory, but we have stated it here specifically for estimation of some regression function f using $\hat{f}$ at some point x.

$$\text{MSE}\left(f(x), \hat{f}(x)\right) = \text{bias}^2\left(\hat{f}(x)\right) + \text{var}\left(\hat{f}(x)\right)$$

In a perfect world, we would be able to find some $\hat{f}$ which is **unbiased**, that is $\text{bias}\left(\hat{f}(x)\right) = 0$, which also has low variance. In practice, this isn't always possible.

It turns out, there is a **bias-variance tradeoff**. That is, often, the more bias in our estimation, the lesser the variance. Similarly, less variance is often accompanied by more bias. Complex models tend to be unbiased, but highly variable. Simple models are often extremely biased, but have low variance.

In the context of regression, models are biased when:

- Parametric: The form of the model does not incorporate all the necessary variables, or the form of the relationship is too simple. For example, a parametric model assumes a linear relationship, but the true relationship is quadratic.
- Non-parametric: The model provides too much smoothing.

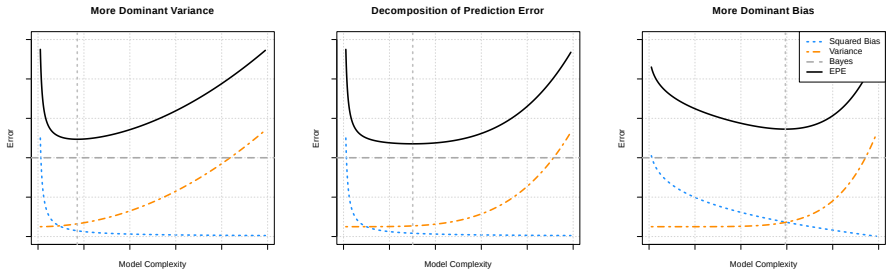
In the context of regression, models are variable when:

- Parametric: The form of the model incorporates too many variables, or the form of the relationship is too complex. For example, a parametric model assumes a cubic relationship, but the true relationship is linear.
- Non-parametric: The model does not provide enough smoothing. It is very, "wiggly."

So for us, to select a model that appropriately balances the tradeoff between bias and variance, and thus minimizes the reducible error, we need to select a model of the appropriate complexity for the data.

Recall that when fitting models, we've seen that train RMSE decreases as model complexity is increasing. (Technically it is non-increasing.) For test RMSE, we expect to see a U-shaped curve. Importantly, test RMSE decreases, until a certain complexity, then begins to increase.

Now we can understand why this is happening. The expected test RMSE is essentially the expected prediction error, which we now known decomposes into (squared) bias, variance, and the irreducible Bayes error. The following plots show three examples of this.



The three plots show three examples of the bias-variance tradeoff. In the left panel, the variance influences the expected prediction error more than the bias. In the right panel, the opposite is true. The middle panel is somewhat neutral. In all cases, the difference between the Bayes error (the horizontal dashed grey line) and the expected prediction error (the solid black curve) is exactly the mean squared error, which is the sum of the squared bias (blue curve) and variance (orange curve). The vertical line indicates the complexity that minimizes the prediction error.

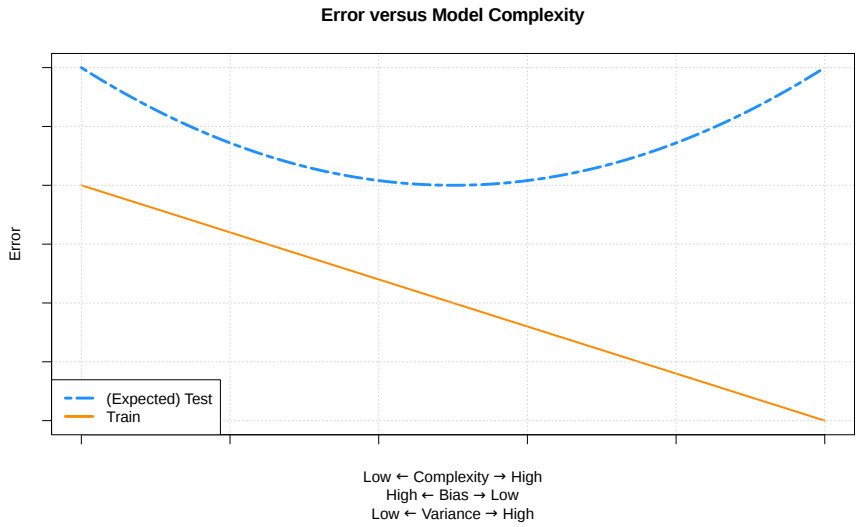
To summarize, if we assume that irreducible error can be written as

$$\mathbb{V}[Y \mid X = x] = \sigma^2$$

then we can write the full decomposition of the expected prediction error of predicting Y using $\hat{f}$ when $X = x$ as

$$\text{EPE}\left(Y, \hat{f}(x)\right) = \underbrace{\text{bias}^2\left(\hat{f}(x)\right) + \text{var}\left(\hat{f}(x)\right)}_{\text{reducible error}} + \sigma^2.$$

As model complexity increases, bias decreases, while variance increases. By understanding the tradeoff between bias and variance, we can manipulate model complexity to find a model that well predict well on unseen observations.



5.3 Simulation

We will illustrate these decompositions, most importantly the bias-variance tradeoff, through simulation. Suppose we would like to train a model to learn the true regression function function $f(x) = x^2$.

```
f = function(x) {
  x ^ 2
}
```

More specifically, we'd like to predict an observation, Y, given that $X = x$ by using $\hat{f}(x)$ where

$$\mathbb{E}[Y \mid X = x] = f(x) = x^2$$

and

$$\mathbb{V}[Y \mid X = x] = \sigma^2.$$

Alternatively, we could write this as

$$Y = f(X) + \epsilon$$

where $\mathbb{E}[\epsilon] = 0$ and $\mathbb{V}[\epsilon] = \sigma^2$. In this formulation, we call $f(X)$ the **signal** and ϵ the **noise**.

To carry out a concrete simulation example, we need to fully specify the data generating process. We do so with the following R code.

```
gen_sim_data = function(f, sample_size = 100) {
  x = runif(n = sample_size, min = 0, max = 1)
  y = rnorm(n = sample_size, mean = f(x), sd = 0.3)
  data.frame(x, y)
}
```

Also note that if you prefer to think of this situation using the $Y = f(X) + \epsilon$ formulation, the following code represents the same data generating process.

```
gen_sim_data = function(f, sample_size = 100) {
  x = runif(n = sample_size, min = 0, max = 1)
  eps = rnorm(n = sample_size, mean = 0, sd = 0.75)
  y = f(x) + eps
  data.frame(x, y)
}
```

To completely specify the data generating process, we have made more model assumptions than simply $\mathbb{E}[Y \mid X = x] = x^2$ and $\mathbb{V}[Y \mid X = x] = \sigma^2$. In particular,

- The x_i in $\mathcal{D}$ are sampled from a uniform distribution over $[0, 1]$.
- The x_i and ϵ are independent.
- The y_i in $\mathcal{D}$ are sampled from the conditional normal distribution.

$$Y \mid X \sim N(f(x), \sigma^2)$$

Using this setup, we will generate datasets, $\mathcal{D}$, with a sample size $n = 100$ and fit four models.

$$\begin{aligned}
\texttt{predict(fit0, x)} &= \hat{f}_0(x) = \hat{\beta}_0 \\
\texttt{predict(fit1, x)} &= \hat{f}_1(x) = \hat{\beta}_0 + \hat{\beta}_1 x \\
\texttt{predict(fit2, x)} &= \hat{f}_2(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 \\
\texttt{predict(fit9, x)} &= \hat{f}_9(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \ldots + \hat{\beta}_9 x^9
\end{aligned}$$

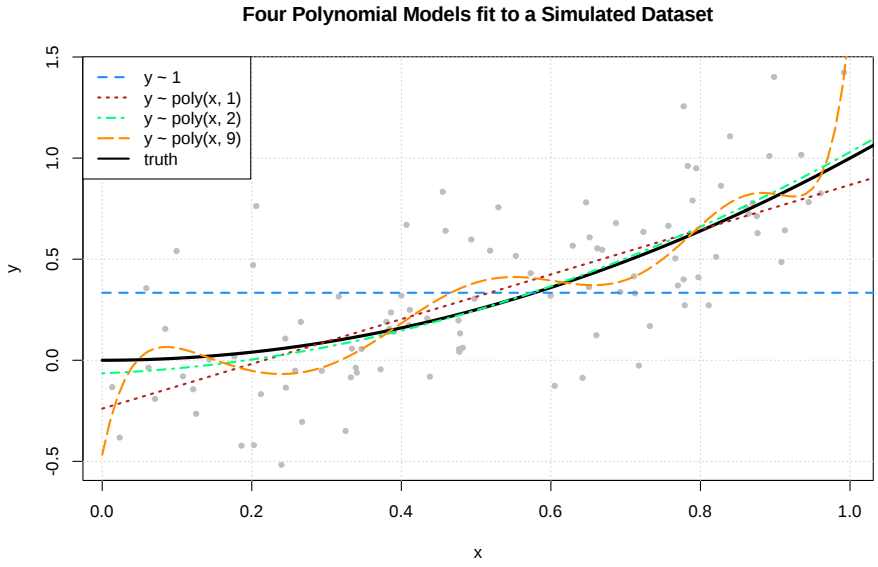
To get a sense of the data and these four models, we generate one simulated dataset, and fit the four models.

```
set.seed(1)
sim_data = gen_sim_data(f)
```

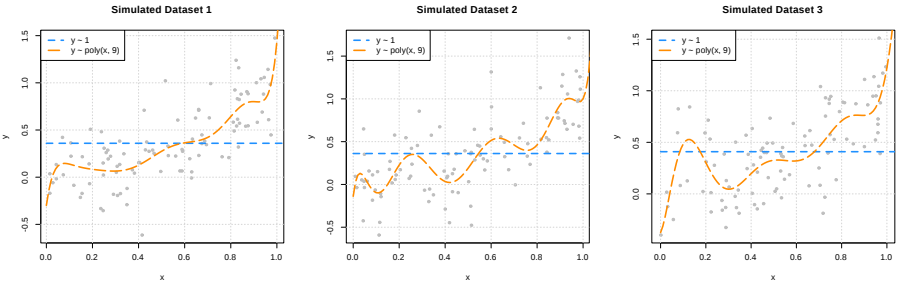
```
fit_0 = lm(y ~ 1,                    data = sim_data)
fit_1 = lm(y ~ poly(x, degree = 1), data = sim_data)
fit_2 = lm(y ~ poly(x, degree = 2), data = sim_data)
fit_9 = lm(y ~ poly(x, degree = 9), data = sim_data)
```

Note that technically we're being lazy and using orthogonal polynomials, but the fitted values are the same, so this makes no difference for our purposes.

Plotting these four trained models, we see that the zero predictor model does very poorly. The first degree model is reasonable, but we can see that the second degree model fits much better. The ninth degree model seem rather wild.



The following three plots were created using three additional simulated datasets. The zero predictor and ninth degree polynomial were fit to each.

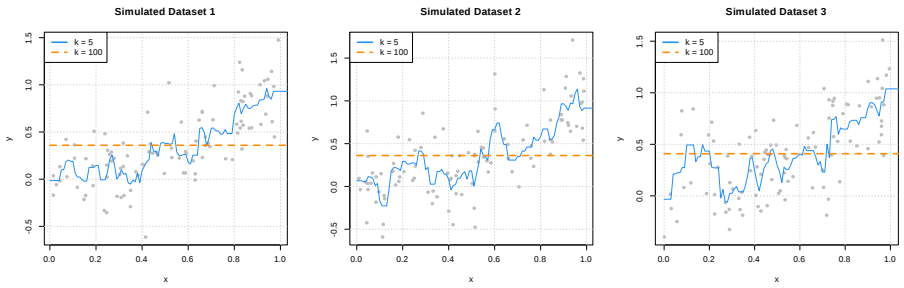


This plot should make clear the difference between the bias and variance of these two models. The zero predictor model is clearly wrong, that is, biased, but nearly the same for each of the datasets, since it has very low variance.

While the ninth degree model doesn't appear to be correct for any of these three

simulations, we'll see that on average it is, and thus is performing unbiased estimation. These plots do however clearly illustrate that the ninth degree polynomial is extremely variable. Each dataset results in a very different fitted model. Correct on average isn't the only goal we're after, since in practice, we'll only have a single dataset. This is why we'd also like our models to exhibit low variance.

We could have also fit k-nearest neighbors models to these three datasets.



Here we see that when $k = 100$ we have a biased model with very low variance. (It's actually the same as the 0 predictor linear model.) When $k = 5$, we again have a highly variable model.

These two sets of plots reinforce our intuition about the bias-variance tradeoff. Complex models (ninth degree polynomial and $k = 5$) are highly variable, and often unbiased. Simple models (zero predictor linear model and $k = 100$) are very biased, but have extremely low variance.

We will now complete a simulation study to understand the relationship between the bias, variance, and mean squared error for the estimates for $f(x)$ given by these four models at the point $x = 0.90$. We use simulation to complete this task, as performing the analytical calculations would prove to be rather tedious and difficult.

```
set.seed(1)
n_sims = 250
n_models = 4
x = data.frame(x = 0.90) # fixed point at which we make predictions
predictions = matrix(0, nrow = n_sims, ncol = n_models)
```

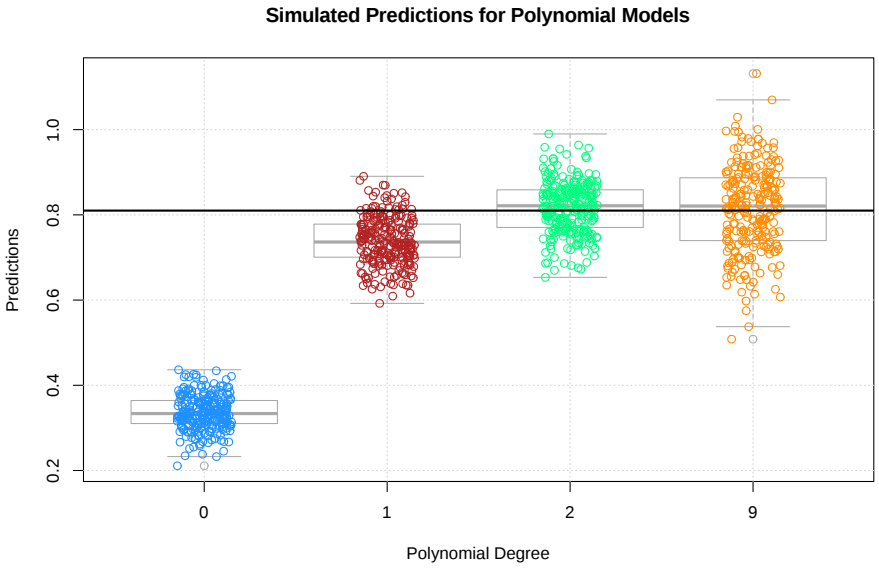
```
for (sim in 1:n_sims) {

  # simulate new, random, training data
  # this is the only random portion of the bias, var, and mse calculations
  # this allows us to calculate the expectation over D
  sim_data = gen_sim_data(f)

  # fit models
```

```
  fit_0 = lm(y ~ 1,                    data = sim_data)
  fit_1 = lm(y ~ poly(x, degree = 1), data = sim_data)
  fit_2 = lm(y ~ poly(x, degree = 2), data = sim_data)
  fit_9 = lm(y ~ poly(x, degree = 9), data = sim_data)

  # get predictions
  predictions[sim, 1] = predict(fit_0, x)
  predictions[sim, 2] = predict(fit_1, x)
  predictions[sim, 3] = predict(fit_2, x)
  predictions[sim, 4] = predict(fit_9, x)
}
```

Note that this is one of many ways we could have accomplished this task using `R`. For example we could have used a combination of `replicate()` and `*apply()` functions. Alternatively, we could have used a `tidyverse` approach, which likely would have used some combination of `dplyr`, `tidyr`, and `purrr`.

Our approach, which would be considered a `base R` approach, was chosen to make it as clear as possible what is being done. The `tidyverse` approach is rapidly gaining popularity in the `R` community, but might make it more difficult to see what is happening here, unless you are already familiar with that approach.

Also of note, while it may seem like the output stored in `predictions` would meet the definition of tidy data given by Hadley Wickham since each row represents a simulation, it actually falls slightly short. For our data to be tidy, a row should store the simulation number, the model, and the resulting prediction. We've actually already aggregated one level above this. Our observational unit is a simulation (with four predictions), but for tidy data, it should be a single prediction. This may be revised by the author later when there are more examples of how to do this from the `R` community.



The above plot shows the predictions for each of the 250 simulations of each of the four models of different polynomial degrees. The truth, $f(x = 0.90) = (0.9)^2 = 0.81$, is given by the solid black horizontal line.

Two things are immediately clear:

- As complexity *increases*, **bias decreases**. (The mean of a model's predictions is closer to the truth.)
- As complexity *increases*, **variance increases**. (The variance about the mean of a model's predictions increases.)

The goal of this simulation study is to show that the following holds true for each of the four models.

$$\text{MSE}\left(f(0.90), \hat{f}_k(0.90)\right) = \underbrace{\left(\mathbb{E}\left[\hat{f}_k(0.90)\right] - f(0.90)\right)^2}_{\text{bias}^2\left(\hat{f}_k(0.90)\right)} + \underbrace{\mathbb{E}\left[\left(\hat{f}_k(0.90) - \mathbb{E}\left[\hat{f}_k(0.90)\right]\right)^2\right]}_{\text{var}\left(\hat{f}_k(0.90)\right)}$$

We'll use the empirical results of our simulations to estimate these quantities. (Yes, we're using estimation to justify facts about estimation.) Note that we've actually used a rather small number of simulations. In practice we should use more, but for the sake of computation time, we've performed just enough simulations to obtain the desired results. (Since we're estimating estimation, the bigger the sample size, the better.)

To estimate the mean squared error of our predictions, we'll use

$$\widehat{\text{MSE}}\left(f(0.90), \hat{f}_k(0.90)\right) = \frac{1}{n_{\texttt{sims}}} \sum_{i=1}^{n_{\texttt{sims}}} \left(f(0.90) - \hat{f}_k(0.90)\right)^2$$

We also write an accompanying `R` function.

```
get_mse = function(truth, estimate) {
  mean((estimate - truth) ^ 2)
}
```

Similarly, for the bias of our predictions we use,

$$\widehat{\text{bias}}\left(\hat{f}(0.90)\right) = \frac{1}{n_{\texttt{sims}}} \sum_{i=1}^{n_{\texttt{sims}}} \left(\hat{f}_k(0.90)\right) - f(0.90)$$

And again, we write an accompanying `R` function.

```
get_bias = function(estimate, truth) {
  mean(estimate) - truth
}
```

Lastly, for the variance of our predictions we have

$$\widehat{\text{var}}\left(\hat{f}(0.90)\right) = \frac{1}{n_{\texttt{sims}}} \sum_{i=1}^{n_{\texttt{sims}}} \left(\hat{f}_k(0.90) - \frac{1}{n_{\texttt{sims}}} \sum_{i=1}^{n_{\texttt{sims}}} \hat{f}_k(0.90)\right)^2$$

While there is already `R` function for variance, the following is more appropriate in this situation.

```
get_var = function(estimate) {
  mean((estimate - mean(estimate)) ^ 2)
}
```

To quickly obtain these results for each of the four models, we utilize the `apply()` function.

```
bias = apply(predictions, 2, get_bias, truth = f(x = 0.90))
variance = apply(predictions, 2, get_var)
mse = apply(predictions, 2, get_mse, truth = f(x = 0.90))
```

We summarize these results in the following table.

Degree	Mean Squared Error	Bias Squared	Variance
0	0.22643	0.22476	0.00167
1	0.00829	0.00508	0.00322
2	0.00387	0.00005	0.00381
9	0.01019	0.00002	0.01017

A number of things to notice here:

- We use squared bias in this table. Since bias can be positive or negative, squared bias is more useful for observing the trend as complexity increases.
- The squared bias trend which we see here is **decreasing** as complexity increases, which we expect to see in general.
- The exact opposite is true of variance. As model complexity increases, variance **increases**.
- The mean squared error, which is a function of the bias and variance, decreases, then increases. This is a result of the bias-variance tradeoff. We can decrease bias, by increasing variance. Or, we can decrease variance by increasing bias. By striking the correct balance, we can find a good mean squared error!

We can check for these trends with the `diff()` function in `R`.

```
all(diff(bias ^ 2) < 0)
```

```
## [1] TRUE
```

```
all(diff(variance) > 0)
```

```
## [1] TRUE
```

```
diff(mse) < 0
```

```
##     1     2     9
##  TRUE  TRUE FALSE
```

The models with polynomial degrees 2 and 9 are both essentially unbiased. We see some bias here as a result of using simulation. If we increased the number of simulations, we would see both biases go down. Since they are both unbiased, the model with degree 2 outperforms the model with degree 9 due to its smaller variance.

Models with degree 0 and 1 are biased because they assume the wrong form of the regression function. While the degree 9 model does this as well, it does include all the necessary polynomial degrees.

$$\hat{f}_9(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \ldots + \hat{\beta}_9 x^9$$

Then, since least squares estimation is unbiased, importantly,

$$\mathbb{E}[\hat{\beta}_d] = \beta_d = 0$$

for $d = 3, 4, \ldots 9$, we have

$$\mathbb{E}\left[\hat{f}_9(x)\right] = \beta_0 + \beta_1 x + \beta_2 x^2$$

Now we can finally verify the bias-variance decomposition.

```
bias ^ 2 + variance == mse
```

```
##     0     1     2     9
## FALSE FALSE FALSE  TRUE
```

But wait, this says it isn't true, except for the degree 9 model? It turns out, this is simply a computational issue. If we allow for some very small error tolerance, we see that the bias-variance decomposition is indeed true for predictions from these for models.

```
all.equal(bias ^ 2 + variance, mse)
```

```
## [1] TRUE
```

See `?all.equal()` for details.

So far, we've focused our efforts on looking at the mean squared error of estimating $f(0.90)$ using $\hat{f}(0.90)$. We could also look at the expected prediction error of using $\hat{f}(X)$ when $X = 0.90$ to estimate Y.

$$\text{EPE}\left(Y, \hat{f}_k(0.90)\right) = \mathbb{E}_{Y|X,\mathcal{D}}\left[\left(Y - \hat{f}_k(X)\right)^2 \mid X = 0.90\right]$$

We can estimate this quantity for each of the four models using the simulation study we already performed.

```
get_epe = function(realized, estimate) {
  mean((realized - estimate) ^ 2)
}
```

```
y = rnorm(n = nrow(predictions), mean = f(x = 0.9), sd = 0.3)
epe = apply(predictions, 2, get_epe, realized = y)
epe
```

```
##         0         1         2         9
## 0.3180470 0.1104055 0.1095955 0.1205570
```

What about the unconditional expected prediction error. That is, for any X, not just 0.90. Specifically, the expected prediction error of estimating Y using $\hat{f}(X)$. The following (new) simulation study provides an estimate of

$$\text{EPE}\left(Y, \hat{f}_k(X)\right) = \mathbb{E}_{X,Y,\mathcal{D}}\left[\left(Y - \hat{f}_k(X)\right)^2\right]$$

for the quadratic model, that is $k = 2$ as we have defined k.

```
set.seed(1)
n_sims = 1000
```

```
X = runif(n = n_sims, min = 0, max = 1)
Y = rnorm(n = n_sims, mean = f(X), sd = 0.3)

f_hat_X = rep(0, length(X))

for (i in seq_along(X)) {
  sim_data = gen_sim_data(f)
  fit_2 = lm(y ~ poly(x, degree = 2), data = sim_data)
  f_hat_X[i] = predict(fit_2, newdata = data.frame(x = X[i]))
}

mean((Y - f_hat_X) ^ 2)
```

```
## [1] 0.09997319
```

Note that in practice, we should use many more simulations in this study.

5.4 Estimating Expected Prediction Error

While previously, we only decomposed the expected prediction error conditionally, a similar argument holds unconditionally.

Assuming

$$\mathbb{V}[Y \mid X = x] = \sigma^2.$$

we have

$$\text{EPE}\left(Y, \hat{f}(X)\right) = \mathbb{E}_{X,Y,\mathcal{D}}\left[(Y - \hat{f}(X))^2\right] = \underbrace{\mathbb{E}_X\left[\text{bias}^2\left(\hat{f}(X)\right)\right] + \mathbb{E}_X\left[\text{var}\left(\hat{f}(X)\right)\right]}_{\text{reducible error}} + \sigma^2$$

Lastly, we note that if

$$\mathcal{D} = \mathcal{D}_{\texttt{trn}} \cup \mathcal{D}_{\texttt{tst}} = (x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, \ i = 1, 2, \ldots n$$

where

$$\mathcal{D}_{\texttt{trn}} = (x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, \ i \in \texttt{trn}$$

and

$$\mathcal{D}_{\texttt{tst}} = (x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, \ i \in \texttt{tst}$$

Then, if we use $\mathcal{D}_{\texttt{trn}}$ to fit (train) a model, we can use the test mean squared error

$$\sum_{i \in \texttt{tst}} \left(y_i - \hat{f}(x_i)\right)^2$$

as an estimate of

$$\mathbb{E}_{X,Y,\mathcal{D}}\left[(Y - \hat{f}(X))^2\right]$$

the expected prediction error. (In practice we prefer RMSE to MSE for comparing models and reporting because of the units.)

How good is this estimate? Well, if $\mathcal{D}$ is a random sample from (X, Y), and `tst` are randomly sampled observations randomly sampled from $i = 1, 2, \ldots, n$, then it is a reasonable estimate. However, it is rather variable due to the randomness of selecting the observations for the test set. How variable? It turns out, pretty variable. While it's a justified estimate, eventually we'll introduce cross-validation as a procedure better suited to performing this estimation to select a model.

5.5 Reproducibility

The R Markdown file for this chapter can be found **here**. The file was created using `R` version 3.6.1.

Chapter 6

Classification

6.1 STAT 432 Materials

- **Slides** | Classification: Introduction
- **Code** | Some Classification Code
- **Slides** | Classification: Binary Classification
- **Code** | Some Binary Classification Code
- **Slides** | Classification: Nonparametric Classification
- **Reading** | STAT 420: Logistic Regression
- **Slides** | Classification: Logistic Regression

6.2 Bayes Classifier

- TODO: Not the same as naïve Bayes classifier

$$C^B(x) = \underset{k \in \{1,2,\ldots K\}}{\text{argmax}} \; P\left[Y = k \mid X = x\right]$$

6.2.1 Bayes Error Rate

$$1 - \mathbb{E}\left[\max_k P[Y = k \mid X]\right]$$

- TODO: https://topepo.github.io/caret/visualizations.html
- TODO: https://en.wikipedia.org/wiki/Confusion_matrix

- TODO: https://en.wikipedia.org/wiki/Matthews_correlation_coefficient
- TODO: https://people.inf.elte.hu/kiss/11dwhdm/roc.pdf
- TODO: https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf
- TODO: http://www.oranlooney.com/post/viz-tsne/
- TODO: https://web.expasy.org/pROC/
- TODO: https://bmcbioinformatics.biomedcentral.com/track/pdf/10.1186/1471-2105-12-77
- TODO: https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- TODO: https://papers.nips.cc/paper/2020-on-discriminative-vs-generative-classifiers-a-compariso pdf
- https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.751&rep=rep1&type=pdf
- https://www.cs.ubc.ca/~murphyk/Teaching/CS340-Fall06/lectures/naiveBayes.pdf
- http://www.stat.cmu.edu/~ryantibs/statml/lectures/linearclassification.pdf
- https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

```
library(tibble)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(rpart)
library(nnet)
```

```
gen_data = function(n = 100) {
  x = sample(c(0, 1), prob = c(0.4, 0.6), size = n, replace = TRUE)
  y = ifelse(test = {x == 0},
             yes = sample(c("A", "B", "C"), size = n, prob = c(0.25, 0.50, 0.25), repla
             no = sample(c("A", "B", "C"), size = n, prob = c(0.1, 0.1, 0.4) / 0.6, rep

  tibble(x = x, y = factor(y))
}

test_cases = tibble(x = c(0, 1))

set.seed(42)
some_data = gen_data()

predict(knn3(y ~ x, data = some_data), test_cases)
```

```
##               A          B         C
## [1,] 0.2608696 0.39130435 0.3478261
## [2,] 0.1481481 0.07407407 0.7777778
```

```
predict(rpart(y ~ x, data = some_data), test_cases)
```

```
##           A          B         C
## 1 0.2608696 0.39130435 0.3478261
## 2 0.1481481 0.07407407 0.7777778
```

```
predict(nnet(y ~ x, data = some_data, size = 0, skip = TRUE, trace = FALSE), test_cases)
```

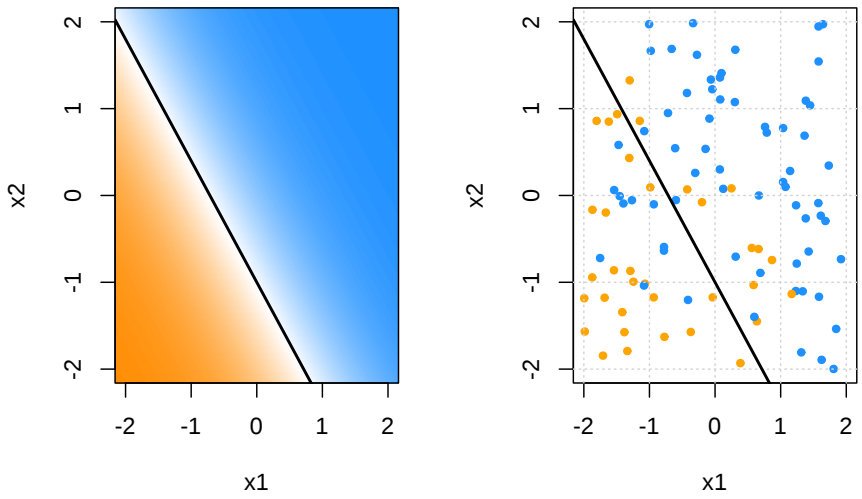
```
##           A          B         C
## 1 0.2608693 0.39130387 0.3478268
## 2 0.1481479 0.07407422 0.7777779
```

6.3 Modeling

6.3.1 Linear Models

```
sim_2d_logistic = function(beta_0, beta_1, beta_2, n) {

  par(mfrow = c(1, 2))

  prob_plane = as_tibble(expand.grid(x1 = -220:220 / 100,
                                     x2 = -220:220 / 100))
  prob_plane$p = with(prob_plane,
                      boot::inv.logit(beta_0 + beta_1 * x1 + beta_2 * x2))

  do_to_db = colorRampPalette(c('darkorange', "white", 'dodgerblue'))

  plot(x2 ~ x1, data = prob_plane,
       col = do_to_db(100)[as.numeric(cut(prob_plane$p,
                                          seq(0, 1, length.out = 101)))],
       xlim = c(-2, 2), ylim = c(-2, 2), pch = 20)
  abline(-beta_0 / beta_2, -beta_1 / beta_2, col = "black", lwd = 2)

  x1 = runif(n = n, -2, 2)
  x2 = runif(n = n, -2, 2)
  y = rbinom(n = n, size = 1, prob = boot::inv.logit(beta_0 + beta_1 * x1 + beta_2 * x2))
  y = ifelse(y == 1, "dodgerblue", "orange")
  asdf = tibble(x1, x2, y)

  plot(x2 ~ x1, data = asdf, col = y, xlim = c(-2, 2), ylim = c(-2, 2), pch = 20)
  grid()
  abline(-beta_0 / beta_2, -beta_1 / beta_2, col = "black", lwd = 2)

}
```

```
sim_2d_logistic(beta_0 = 2 * 0.5, beta_1 = 2* 0.7, beta_2 = 2* 0.5, n = 100)
```



6.3.2 k-Nearest Neighbors

6.3.3 Decision Trees

Chapter 7

Resampling

- TODO: https://github.com/topepo/caret/issues/70
- TODO: https://stats.stackexchange.com/questions/266225/step-by-step-explanation-of-k-fold-cross-validation
- TODO: https://weina.me/nested-cross-validation/
- **Code** | Some Resampling Code

Chapter 8

Supervised Learning

- TODO: write an overview / review of the previous two chapters
- TODO: do two analyses?

Appendix A

Probability

- TODO: Note! This is copy-pasted from R4SL.

We give a very brief review of some necessary probability concepts. As the treatment is less than complete, a list of references is given at the end of the chapter. For example, we ignore the usual recap of basic set theory and omit proofs and examples.

A.1 Probability Models

When discussing probability models, we speak of random **experiments** that produce one of a number of possible **outcomes**.

A **probability model** that describes the uncertainty of an experiment consists of two elements:

- The **sample space**, often denoted as Ω, which is a set that contains all possible outcomes.
- A **probability function** that assigns to an event A a nonnegative number, $P[A]$, that represents how likely it is that event A occurs as a result of the experiment.

We call $P[A]$ the **probability** of event A. An **event** A could be any subset of the sample space, not necessarily a single possible outcome. The probability law must follow a number of rules, which are the result of a set of axioms that we introduce now.

A.2 Probability Axioms

Given a sample space Ω for a particular experiment, the **probability function** associated with the experiment must satisfy the following axioms.

1. *Nonnegativity*: $P[A] \geq 0$ for any event $A \subset \Omega$.
2. *Normalization*: $P[\Omega] = 1$. That is, the probability of the entire space is 1.
3. *Additivity*: For mutually exclusive events $E_1, E_2, \ldots$

$$P\left[\bigcup_{i=1}^{\infty} E_i\right] = \sum_{i=1}^{\infty} P[E_i]$$

Using these axioms, many additional probability rules can easily be derived.

A.3 Probability Rules

Given an event A, and its complement, A^c, that is, the outcomes in Ω which are not in A, we have the **complement rule**:

$$P[A^c] = 1 - P[A]$$

In general, for two events A and B, we have the **addition rule**:

$$P[A \cup B] = P[A] + P[B] - P[A \cap B]$$

If A and B are also *disjoint*, then we have:

$$P[A \cup B] = P[A] + P[B]$$

If we have n mutually exclusive events, $E_1, E_2, \ldots E_n$, then we have:

$$P\left[\bigcup_{i=1}^{n} E_i\right] = \sum_{i=1}^{n} P[E_i]$$

Often, we would like to understand the probability of an event A, given some information about the outcome of event B. In that case, we have the **conditional probability rule** provided $P[B] > 0$.

$$P[A \mid B] = \frac{P[A \cap B]}{P[B]}$$

Rearranging the conditional probability rule, we obtain the **multiplication rule**:

$$P[A \cap B] = P[B] \cdot P[A \mid B]\cdot$$

For a number of events $E_1, E_2, \ldots E_n$, the multiplication rule can be expanded into the **chain rule**:

$$P\left[\bigcap_{i=1}^{n} E_i\right] = P[E_1] \cdot P[E_2 \mid E_1] \cdot P[E_3 \mid E_1 \cap E_2] \cdots P\left[E_n \mid \bigcap_{i=1}^{n-1} E_i\right]$$

Define a **partition** of a sample space Ω to be a set of disjoint events $A_1, A_2, \ldots, A_n$ whose union is the sample space Ω. That is

$$A_i \cap A_j = \emptyset$$

for all $i \neq j$, and

$$\bigcup_{i=1}^{n} A_i = \Omega.$$

Now, let $A_1, A_2, \ldots, A_n$ form a partition of the sample space where $P[A_i] > 0$ for all i. Then for any event B with $P[B] > 0$ we have **Bayes' Rule**:

$$P[A_i|B] = \frac{P[A_i]P[B|A_i]}{P[B]} = \frac{P[A_i]P[B|A_i]}{\sum_{i=1}^{n} P[A_i]P[B|A_i]}$$

The denominator of the latter equality is often called the **law of total probability**:

$$P[B] = \sum_{i=1}^{n} P[A_i]P[B|A_i]$$

Two events A and B are said to be **independent** if they satisfy

$$P[A \cap B] = P[A] \cdot P[B]$$

This becomes the new multiplication rule for independent events.

A collection of events $E_1, E_2, \ldots E_n$ is said to be independent if

$$P\left[\bigcap_{i\in S} E_i\right] = \prod_{i\in S} P[E_i]$$

for every subset S of $\{1, 2, \ldots n\}$.

If this is the case, then the chain rule is greatly simplified to:

$$P\left[\bigcap_{i=1}^{n} E_i\right] = \prod_{i=1}^{n} P[E_i]$$

A.4 Random Variables

A **random variable** is simply a *function* which maps outcomes in the sample space to real numbers.

A.4.1 Distributions

We often talk about the **distribution** of a random variable, which can be thought of as:

$$\text{distribution} = \text{list of possible } \textbf{values} + \text{associated } \textbf{probabilities}$$

This is not a strict mathematical definition, but is useful for conveying the idea.

If the possible values of a random variables are *discrete*, it is called a *discrete random variable*. If the possible values of a random variables are *continuous*, it is called a *continuous random variable*.

A.4.2 Discrete Random Variables

The distribution of a discrete random variable X is most often specified by a list of possible values and a probability **mass** function, $p(x)$. The mass function directly gives probabilities, that is,

$$p(x) = p_X(x) = P[X = x].$$

Note we almost always drop the subscript from the more correct $p_X(x)$ and simply refer to $p(x)$. The relevant random variable is discerned from context

The most common example of a discrete random variable is a **binomial** random variable. The mass function of a binomial random variable X, is given by

$$p(x|n,p) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, 1, \ldots, n, \ n \in \mathbb{N}, \ 0 < p < 1.$$

This line conveys a large amount of information.

- The function $p(x|n,p)$ is the mass function. It is a function of x, the possible values of the random variable X. It is conditional on the **parameters** n and p. Different values of these parameters specify different binomial distributions.
- $x = 0, 1, \ldots, n$ indicates the **sample space**, that is, the possible values of the random variable.
- $n \in \mathbb{N}$ and $0 < p < 1$ specify the **parameter spaces**. These are the possible values of the parameters that give a valid binomial distribution.

Often all of this information is simply encoded by writing

$$X \sim \text{bin}(n,p).$$

A.4.3 Continuous Random Variables

The distribution of a continuous random variable X is most often specified by a set of possible values and a probability **density** function, $f(x)$. (A cumulative density or moment generating function would also suffice.)

The probability of the event $a < X < b$ is calculated as

$$P[a < X < b] = \int_a^b f(x)dx.$$

Note that densities are **not** probabilities.

The most common example of a continuous random variable is a **normal** random variable. The density of a normal random variable X, is given by

$$f(x|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left[\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right], \quad -\infty < x < \infty, \ -\infty < \mu < \infty, \ \sigma > 0.$$

- The function $f(x|\mu,\sigma^2)$ is the density function. It is a function of x, the possible values of the random variable X. It is conditional on the **paramters** μ and σ^2. Different values of these parameters specify different normal distributions.
- $-\infty < x < \infty$ indicates the sample space. In this case, the random variable may take any value on the real line.

- $-\infty < \mu < \infty$ and $\sigma > 0$ specify the parameter space. These are the possible values of the parameters that give a valid normal distribution.

Often all of this information is simply encoded by writing

$$X \sim N(\mu, \sigma^2)$$

A.4.4 Several Random Variables

Consider two random variables X and Y. We say they are independent if

$$f(x, y) = f(x) \cdot f(y)$$

for all x and y. Here $f(x, y)$ is the **joint** density (mass) function of X and Y. We call $f(x)$ the **marginal** density (mass) function of X. Then $f(y)$ the marginal density (mass) function of Y. The joint density (mass) function $f(x, y)$ together with the possible (x, y) values specify the joint distribution of X and Y.

Similar notions exist for more than two variables.

A.5 Expectations

For discrete random variables, we define the **expectation** of the function of a random variable X as follows.

$$\mathbb{E}[g(X)] \triangleq \sum_x g(x)p(x)$$

For continuous random variables we have a similar definition.

$$\mathbb{E}[g(X)] \triangleq \int g(x)f(x)dx$$

For specific functions g, expectations are given names.

The **mean** of a random variable X is given by

$$\mu_X = \text{mean}[X] \triangleq \mathbb{E}[X].$$

So for a discrete random variable, we would have

$$\text{mean}[X] = \sum_x x \cdot p(x)$$

For a continuous random variable we would simply replace the sum by an integral.

The **variance** of a random variable X is given by

$$\sigma_X^2 = \text{var}[X] \triangleq \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2.$$

The **standard deviation** of a random variable X is given by

$$\sigma_X = \text{sd}[X] \triangleq \sqrt{\sigma_X^2} = \sqrt{\text{var}[X]}.$$

The **covariance** of random variables X and Y is given by

$$\text{cov}[X, Y] \triangleq \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X] \cdot \mathbb{E}[Y].$$

A.6 Likelihood

Consider n iid random variables $X_1, X_2, \ldots X_n$. We can then write their **likelihood** as

$$\mathcal{L}(\theta \mid x_1, x_2, \ldots x_n) = \prod_{i=i}^{n} f(x_i; \theta)$$

where $f(x_i; \theta)$ is the density (or mass) function of random variable X_i evaluated at x_i with parameter θ.

Whereas a probability is a function of a possible observed value given a particular parameter value, a likelihood is the opposite. It is a function of a possible parameter value given observed data.

Maximumizing likelihood is a common techinque for fitting a model to data.

A.7 Videos

The YouTube channel mathematicalmonk has a great Probability Primer playlist containing lectures on many fundamental probability concepts. Some of the more important concepts are covered in the following videos:

- Conditional Probability
- Independence
- More Independence
- Bayes Rule

A.8 References

Any of the following are either dedicated to, or contain a good coverage of the details of the topics above.

- Probability Texts
  - Introduction to Probability by Dimitri P. Bertsekas and John N. Tsitsiklis
  - A First Course in Probability by Sheldon Ross
- Machine Learning Texts with Probability Focus
  - Probability for Statistics and Machine Learning by Anirban DasGupta
  - Machine Learning: A Probabilistic Perspective by Kevin P. Murphy
- Statistics Texts with Introduction to Probability
  - Probability and Statistical Inference by Robert V. Hogg, Elliot Tanis, and Dale Zimmerman
  - Introduction to Mathematical Statistics by Robert V. Hogg, Joseph McKean, and Allen T. Craig